



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

1984

Implementation of a Zilog Z-80 base
realization library for the computer system
design environment

Smith, Theodore John

Monterey, California. Naval Postgraduate School

<http://hdl.handle.net/10945/19374>

Downloaded from NPS Archive: Calhoun



Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

IMPLEMENTATION OF A ZILOG Z-80 BASE
REALIZATION LIBRARY FOR THE COMPUTER
SYSTEMS DESIGN ENVIRONMENT

by

Theodore John Smith, Jr.

March 1984

Thesis Advisor:

Alan A. Ross

Approved for Public Release; Distribution Unlimited

T218031

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Implementation of a Zilog Z-80 Based Realization Library for the Computer Systems Design Environment		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis March, 1984
7. AUTHOR(s) Theodore John Smith, Jr.		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE March, 1984
		13. NUMBER OF PAGES 106
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for Public Release; Distribution Unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer aided design, CSDL, Z-80, realization, primitive monitor		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This thesis develops a Zilog-80 based realization library for use in the automated design of microprocessor based control systems. The library is designed around Standard Bus boards. This bus is supported by a number of manufacturers for use in breadboard construction, through a number of standard cards. The library of primitives developed implement the construction used in Computer System Design Language (CSDL) for (Continued)		

ABSTRACT (Continued)

the Z-80 cpu. CSDL is a high level language that allows the specifications of tasks and procedures, and their time constraints. Use of the design system and this library can enable Z-80 based prototype controllers to be quickly designed and built.



Approved for public release; distribution unlimited

Implementation of a Zilog Z-80 Base Realization Library for
the Computer Systems Design Environment

by

Theodore John Smith Jr.
Major, United States Army
B.S., Loyola University, 1970

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1984

ABSTRACT

This thesis develops a Zilog-80 based realization library for use in the automated design of microprocessor based control systems. The library is designed around Standard Bus boards. This bus is supported by a number of manufacturers for use in breadboard construction, through a number of standard cards. The library of primitives developed implement the constructions used in Computer System Design Language (CSDL) for the Z-80 cpu. CSDL is a high level language that allows the specifications of tasks and procedures, and their time constraints. Use of the design system and this library can enable Z-80 based prototype controllers to be quickly designed and built.

TABLE OF CONTENTS

I.	INTRODUCTION-----	9
II.	BACKGROUND-----	13
III.	DESIGN-----	19
	A. THE CURRENT DESIGN SYSTEM-----	19
	B. CURRENT PROBLEMS WITH THE INTEL 8080 LIBRARY-----	21
	C. METHODOLOGY-----	23
IV.	IMPLEMENTATION-----	25
	A. MONITOR-----	25
	B. ARITHMETIC-----	29
	C. CONTROL STRUCTURES-----	31
	D. INTERRUPTS-----	33
	E. INPUT/OUTPUT DEVICES-----	36
	1. Onboard Cpu Three Channel Counter Timer Chip-----	37
	2. 8 Bit Analog To Digital Conversion Board-----	38
	3. 8 Bit Digital To Analog Conversion Board-----	39
	4. 64 Port 8 Bit Standard Bus To Digital I/O Bus-----	39
	5. Keyboard/Display Card-----	40
	6. Dual UART Board-----	41
F.	TESTING-----	41

G.	FORMATTING OF PRIMITIVES-----	42
1.	Pollock Fixit-----	45
2.	Ross's Newcsdl-----	46
3.	Walden's Data Base Input-----	48
V.	RESULTS AND CONCLUSIONS-----	49
	APPENDIX A - PRIMITIVE TITLE INDEX-----	51
	APPENDIX B - GLOBAL VARIABLE LISTING-----	57
	APPENDIX C - ZILOG-80 REALIZATION LISTING-----	60
	LIST OF REFERENCES-----	104
	INITIAL DISTRIBUTION LIST-----	106

LIST OF FIGURES

1.	Cost Versus Production by Language-----	11
2.	Current Ross Controller Design System-----	20
3.	Monitor for Z-80 Realization-----	27

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Lieutenant Colonel Alan Ross, and second reader, Professor Daniel Dolk, for their assistance and support in this thesis. I thank my wife Celeste and Peter for their support and faith in me.

I. INTRODUCTION

During the past ten years there has been a microelectronics revolution in which ever increasing numbers of functions have been put on a single chip. This has translated into a shift in system costs from capital to labor. Hardware is no longer the dominant factor in the cost of a computer system. The rapid drop in the price of general purpose computer chips has also caused a shift in their application. Because of their low cost many of these chips are replacing specialized control hardware.

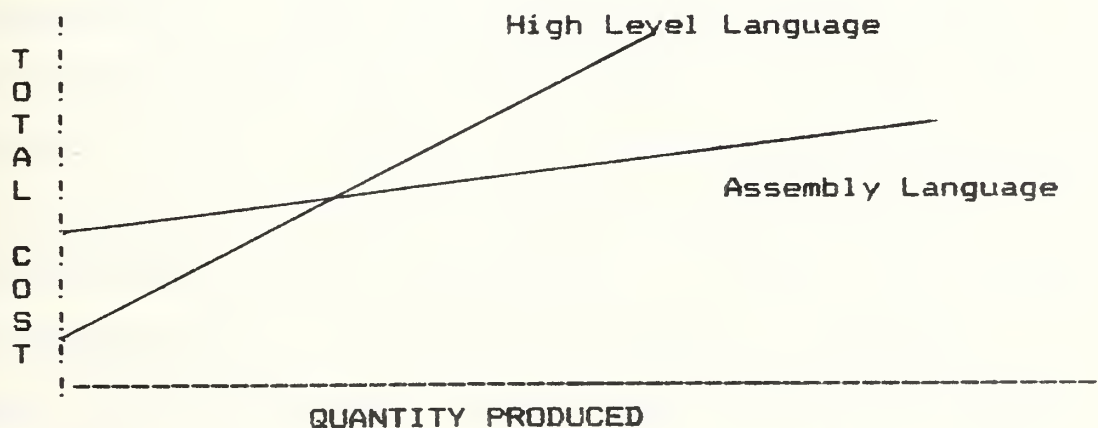
In the past hardware was extremely expensive. Even the design of a simple controller required a large number of components. Many of these simple logic components cost as much as the single chip general purpose controller today. Because hardware was so expensive, controllers were developed to use the minimum amount of hardware possible, at the expense of a great deal of labor. This took the form of engineers trying to minimize the number of gates or logic functions used to implement the controller. The microelectronic industry has now managed to put a functional computer on a chip for the same cost as a few gates ten years ago.

Two examples of control applications are speed control of a power plant generator and starting control of a gas turbine engine. Control of power plant generators has been done in the past by mechanical controllers. The first example shows the displacement of the mechanical controllers because of the high cost mechanical systems. Even though the mechanical controller was expensive, the degree of control was imprecise. Here the driving factor has been the desire to increase the amount of control possible as well as reduce the cost of the controller. The starting of gas turbine engines has been for the most part manual, because of the large number of malfunctions that can occur during starting. Recently, mechanical and digital starting systems have been developed to start gas turbine engines. The second example shows a new application for a controller, primarily based on the reduced cost of digital component.

Unfortunately the very low cost of the microprocessor cpu is a small part of the total system cost. Currently a microprocessor can cost less than four dollars [Ref. 1: p.536], but the cost of a programmer can exceed \$12 per hour [Ref. 2: p.90]. Unless the system being controlled is very costly or the volume produced high, the cost of designing the digital system is not affordable. This cost can also be exacerbated by the choice of programming language. Figure 1 illustrates the costs of programming in a high order language versus assembly language. Note that implicit in this graph is a belief that assembly code will be more

efficient at run time. Current optimizing compilers are becoming more efficient and are approaching the efficiency of assembly code. This has the effect of decreasing the slope of the higher order language total cost line and moving the intersection of the two line to higher production levels. The implication is that except for the most demanding of high production applications, the coding should be done in a high level language.

The second factor affecting the cost of programming is the complexity of the system to be controlled. As the complexity of the control program becomes larger than a single individual can intellectually handle, the program must be subdivided and designed by a team. Currently, hardware is chosen early in the design process and the software is then written for that hardware. Early design errors can cause all subsequent programming to be redone as well as the possibility of having to select different hardware.



Cost Versus Production by language
Figure 1 [Ref. 3: p.432]

The final factor is the speed at which the control must be performed. Some control functions must be accomplished within a critical length of time. This factor competes directly with the previous two factors, cost and complexity. There are two methods to determine if a program is fast enough: program the controller and time the response of the system based on an input or calculate the basic execution time of the instructions in the program. In the first case, exhaustive testing must be used to establish the maximum runtime of the program. The second method is more exact, but is labor intensive. In both cases, the program must be present before the timing can be determined. If the program is too slow it must be rewritten or possibly the hardware must be changed in order to sufficiently increase the system speed.

These problems in design have caused the computer community as well as the controller community to increase the use of simulations and other tools to minimize the risk of these errors. Some have stressed the hardware design, others have stressed the software design, and others have increased the use of models. As costs have decreased, a final group has attempted to design the whole system, including both hardware and software. In this last case we see increased use of prototype development in computer design. In the past this was not done because of the cost of developing the prototype. It is becoming more feasible now because the size and complexity of the final system

makes errors in design too costly to be routinely repaired after the systems are fielded.

This thesis will investigate and develop a library of realizations based on the Zilog Z-80 cpu to support the design system developed by ROSS [Ref. 4: pp.7-8]. Since Ross's original work only contained a single library, this second library will provide the option of building a realization with more than one library. It will be possible to test the design system's capabilities to choose a realization library based on a problem statement. The Z-80 hardware will be provided by the Standard Bus system of prototyping boards, to make it possible to quickly design and reconfigure prototypes. With the completion of these goals it will be possible to construct a working controller using Ross's design system and finally verify the accuracy of this design concept.

II. BACKGROUND

Computer aided design has been an evolving process over the past 20 years, intended to reduce the labor required to engineer a product. This process can be described by tracing several different threads. The first thread to be followed will be the design of controller systems. This implies a ready pool of predesigned hardware. The second

area to be looked at will be the design of hardware. Finally, the design of the complete system will be examined.

Matelan proposed that computer aided design be applied to the design of real time controllers, by adding the use of realization libraries of standard components. He presented a methodology for defining the timing constraints driving a real time controller. This consists of having pairs of contingencies and their associated tasks. By dividing the total problem into paired contingencies and tasks, the individual pairs can be reordered to meet the overall timing constraints.[Ref. 5: pp.17-20]

Ross implemented Matelan's ideas for timing analysis and added the potential for background tasks [Ref. 6: pp.15-22]. The processor chosen for the realization library was the INTEL 8080. This particular processor was chosen because of its availability and low cost. In the course of his thesis, Ross also corrected Matelan's example problem [Ref. 7: p.77]. This came about while trying to debug Ross's timing analyzer. The timing analyzer was correct and Matelan's example was wrong, because Matelan had scheduled a potential second occurrence of a contingency task pair within the time needed for the execution of the first contingency pair. This highlights the need for some automated means of assisting the designer in avoiding similar errors. Ross provided sample executions of the program with accompanying paper hardware realization, however, no actual hardware was built or tested.

Pollock attempted to build an actual hardware realization in the form of a fuel controller for a car. Pollock never did attain this goal in the course of writing his thesis. He did add a variety of additional hardware to the INTEL 8080 realization library to include a floating point chip for floating point operations and transcendental functions. He criticized Ross's FORTRAN implementation of the timing analyzer and functional mapper. He suggested that all of the present programs be scrapped and work begin anew retaining Matelan's theoretical foundations. Much of this criticism appears to be the result of the difficulty in adding additional primitives. This is caused in large part by the column reads performed by the FORTRAN functional mapper on the primitive listing. The structure of the primitive listing itself has pointers that refer to parts of the listing by a relative line displacement that makes changing the primitives tedious. [Ref. 8: p.34]

Manwaring argues for a similar system, independent of Ross and Matelan. His ideas on libraries of implementations for different processors are equivalent to Ross's, but he does not include the selection of the processor by the design system. As in Ross's system he argues for a design system that chooses hardware and the software to run the system, by means of a high level language to state the problem. He also does not consider the analysis of timing in his design system. But he does concede that the compiled code may run too slowly and portions of the software may

have to be manually optimized to meet time constraints. Additionally, his proposed design system does not manipulate the timing problem and realizations to the extent that Ross's does. He proposes generating a compiler error if the desired primitive doesn't exist in a particular processor's library, rather than trying a different hardware realization. He does argue that the rapid generation of programs can make digital controllers available for more applications. [Ref. 9: pp.431-435]

Biehl, also independent of others, presents the system LOGE-MIR to design controllers using microprocessors. The input is in the form of a state table and flow diagram. LOGE-MIR process the state table and flow diagram to obtain an intermediate language. The intermediate language is then manipulated to optimize the following: the sequence in which the condition variables are tested, the assignment of conditional inputs, the control of outputs to ports and the number of jump instructions. This system is in contrast to Ross's in that there are no time constraints, the system merely attempts to make the controller as fast as possible. The ultimate machine code is just a direct translation of the intermediate code to the target controller code, with no guarantee of a specific response. [Ref. 10: pp.328-333]

Finally, Sherlock studied the problem of making entries of the problem into the Problem Statement Analyzer easier. After a lengthy review of human factors engineering, a program was written to make the input of a problem

formulated in Computer System Design Language into Ross's design system easier. [Ref. 11: pp.15-16]

The second thread in the description of automated design tools is the design of the hardware itself. Chu has been working in the area of microcomputer design since 1965. He states that the following need to be described in the language: identification of the selected LSI, MSI, and SSI chips, a plan for the interconnection of these chips and a description of the internal structures and sequences of these chips. He also describes three level of increasing detail in the design process: functional, ideal timing, and real time levels. He proposes that, given a functional description, a computer system could be designed and simulated prior to the actual construction of the system. The system would need a large data base of chip identifications and interconnections to function. He proposes that manufacturers provide disk packs of their chip descriptions in addition to the data books that they now provide. [Ref. 12: pp.45-51]

Heath, Carroll and Cwik describe a modified version of Chu's Computer Design Language that was running at Auburn University. Two new declarations were added to allow the easy implementation of buses and front panel lights. They conclude that the following can be easily tested using simulation: basic system organization, function of some microcode, timing problems, limitations on input, and throughput rate. [Ref. 13: pp.93-108]

Hartenstein and Von Puttkamer describe the language KARL and its associated graphical description ABL. KARL is a Pascal-like language for allowing simulations of a processor at the register transfer level. It is an improvement over CDL in it's original form in that it allows the integration of a graphical description of the design through the use of ABL , a block diagram language. Once inputed, the design can be simulated for correctness and finally the design system can output detailed layout drawing and mask specifications. [Ref. 14: pp.155-160]

The third and final thread is the design of both the hardware and the software. A feasibility study was conducted on the design of an integrated design facility by the US Air Force at the Rome Air Development Center. The initial study was performed by Sperry Univac. The concept included a design facility where total system design alternatives could be emulated for the purpose of providing and evaluating designs prior to actual development. Two important conclusions of the study were: the system could be used for requirements formulation as well as hardware and software design specifications and the present system was inadequate for the analytic determination of operating performance. Further, the study concluded that additional tools should be added to assess operating performance, even though the tools would only provide approximate answers. [Ref. 15: pp.19-20, 253-258]

Having reviewed the current progress in automated controller design, further study on the problems associated with the design system developed by Ross will be done. This is done to limit the scope of the problem to a manageable size.

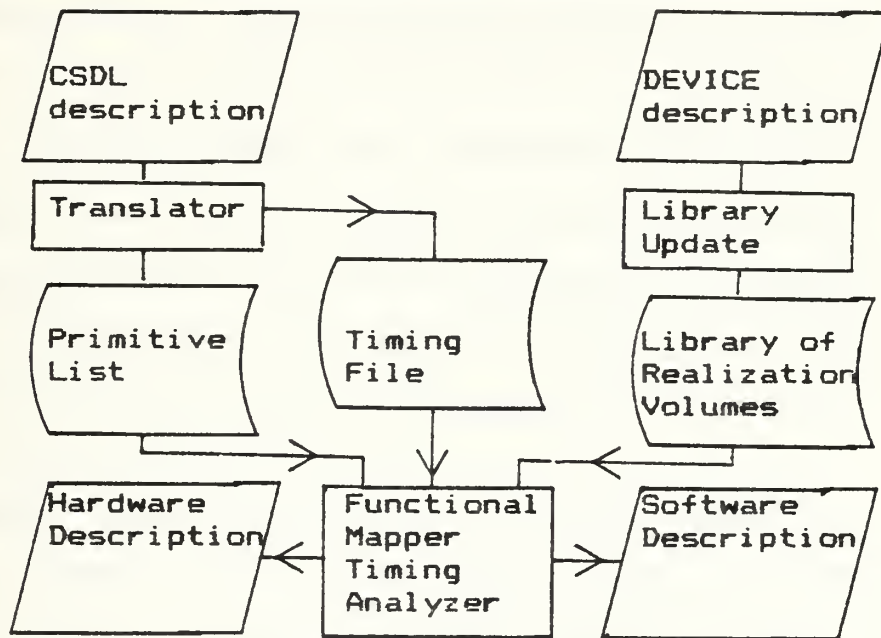
III. DESIGN

A. THE CURRENT DESIGN SYSTEM

The area of study for this thesis is the design system established by Ross. In this system the hardware is selected from predesigned components. The major goal of the system is the rapid design of prototype controllers. The system is not intended to produce a final packaged production design. It will produce a breadboard controller capable of verifying the feasibility of the controller, its desired characteristics, and timing.

The current design system is shown graphically in figure 2. The library of primitives currently contains only an Intel 8080 realization. The implications of this are that if the controller cannot be designed using an Intel 8080 processor, then the system will indicate that the controller is not possible.

The purpose of this research is to add a Zilog Z-80 based realization to that library. Construction of this second library will give the design system an alternative method of realizing a controller. This will also enable further testing to be done on the design system itself.



Current Ross Controller Design System

Figure 2

Another reason for adding the Z-80 realization library is to allow the actual construction of a test controller from the design system. A thesis currently being conducted by Riley will use the design system and the primitives developed in this thesis to construct a generic gas turbine starting controller [Ref. 16]. Since an actual controller will be constructed using the primitives in this thesis, provisions have been added to the primitives to allow ease of debugging and testing.

So far there has been no complete exercise of the design system. Pollock attempted to build a fuel controller for a car but no hardware was built. Heilstedt studied the problem of using the design system to construct digital filters, but also did not build any hardware. Many of his problems related to converting the program to run on a VAX 11/780. A prime purpose of this thesis is to get a working system that can be used for a complete design test in future projects. With this as a goal, the Standard Prolog breadboard system was chosen as a source of hardware. This particular system is available for testing and offers potential for quick prototype assembly.

B. CURRENT PROBLEMS WITH THE INTEL 8080 LIBRARY

The current realization library, based on an Intel 8080, has its origin in Ross's original thesis in 1978. As mentioned before, the library has been modified and expanded by Pollock and Heilstedt, with specific projects in mind. In the six years since the library's inception the cost of hardware has continued to decrease and its power increase. The change in orientation can be seen in the types of primitives put in the realization. Pollock added a great deal of hardware, including a floating point processor, to the 8080 library. The Zilog Z-80 was chosen as a newer chip that could add a higher performance library to the design system. It was also chosen because of its popularity and similarity to the Intel 8080.

The floating point processor added to the Intel 8080 library confuses some of the design issues in using the design system. By adding the floating point chip the cost of the system is greatly increased. Because of their infrequent use, floating point chips do not have a production volume that has allowed the general purpose processing chips to decrease in price. The use of the floating point chip will be eliminated in the Z-80 library. An alternative to this in a hierarchy of processors is a high performance library being designed around the Intel 8088 by CietaI [Ref. 17]. Because of the hardware instructions available, this chip may offer similar performance with reduced cost.

The current Intel 8080 library does not treat negative numbers correctly. The comparison, multiply and division primitives all were written only with positive integers in mind. The Z-80 library will incorporate code that also provides correct operations with 2's complement arithmetic.

The monitor was initially designed to be used without a stack. Because of that the monitor, when calling a routine, used an intermediate table to determine where the task was located. Later Pollock added a stack to the primitive listing, but the monitor structure was never completely modified to take advantage of it.

Finally, there is no protection from the propagation of errors. Should an underflow or overflow occur, the sign of the result will be incorrect and no action is taken to

minimize the effect. In terms of control, it is possible that the item controlled will be directed to perform the exact opposite action from what is required. As an example, if a positive number opens a valve and a negative number closes the valve, then an overflow will cause the valve to be closed at a time when the control program is trying to open the valve wider.

C. METHODOLOGY

To capitalize on the effort that has gone into the Intel 8080 library, each of the primitives will be reviewed. If the primitive is still required, then the initial draft of the Z-80 primitive will be a direct translation of the 8080 code to Z-80 assembly code. Speed improvements will then be attempted using the additional instructions available to the Z-80. If there is a tradeoff to be made in speed or code size, then speed will be chosen. This will eliminate the use of the jump relative instruction, since it is slower but more compact than the jump absolute instruction.

All loop and control structures in each primitive will initially be made with labels to facilitate writing the primitive. Then each primitive will be incrementally tested using a debugger. When the primitive works correctly, the labels will be replaced with relative assembly jumps to insure portability in the actual use of the primitive. Each primitive will then be tested again with a debugger to insure that the primitive still functions correctly. In

testing a primitive all paths through the code will be exercised. To do this numbers from a representative class will be chosen and entered through the debugger.

The floating point processor used in the Intel 8080 library will not be used in the Z80 library. The floating point arithmetic operations will be implemented in software. To insure compatibility with other computers, the IEEE single precision floating point standard will be used as an interface form for external information. Keeping with the spirit of the standard and the intent of the control system, overflows will be given a value of infinity. Though not discussed in the floating point standard, conversions of infinity or numbers larger than the range of an integer value will be converted to the largest integer value representable. This is an attempt to minimize the propagation of opposite control responses.

The completed primitives will be aggregated into a single library file and then processed into a format acceptable to Ross's Design system by use of Pollock's formatting utility. This will allow the primitives to be individually tested. Being able to add a few primitives at a time and then have them formatted by Pollock's formatter program can isolate any problems introduced by the newly added primitives. Using Pollock's formatting program enables a primitive library to use Ross's design program directly, without having to make tedious computations of various pointers. Walden's data base storage [Ref. 18] of

the primitives will not be tested by this library, in order to minimize the amount of change to the design system and because it does not allow the entry of code from a file.

Appendix A lists the names of the primitives and a brief description of their purpose. This list has been compiled to reduce the volume of data that must be scanned to understand the coverage of the library primitives. It should be noted that in contrast to the Intel 8080 library, many of the hardware primitives are eliminated in favor of software primitives. Further, the aggregation of the hardware primitive is now at the board level, rather than the component level.

IV. IMPLEMENTATION

A. MONITOR

The original monitor structure used by Ross and Pollock consists of an infinite loop of contingency task pairs. The order of the pairs is determined by Ross's timing analyzer based on the timing constraints of the various tasks. Control is transferred by the use of an intermediate table which contains the actual address of the task to be executed. When a task is completed, control is then returned to the task loop. A task counter is incremented and the next task is executed via the intermediate table. When all tasks in the loop have been executed once, the task counter is reset to point to the first task in the loop and

the process repeats itself. This structure has its source in the original library developed by Ross which did not have a stack. The lack of a stack prohibited the use of calls and returns for subroutines. Later a stack was added and subroutines used to transfer control to the contingency test, but the monitor was not changed.

The Z80 monitor, shown in Figure 3, consists of a main loop that has a single entry and no exit. Switches cause various submonitors to be executed from the main monitor loop. This was done to minimize testing of conditions that are mutually exclusive. Timing analysis is done on the submonitors plus the time to execute the main monitor loop with all switches false except one. All tasks, conditions and procedures are executed as subroutine calls from the polling loop in the submonitor. Initialization is handled as a submonitor without any timing constraint. The initialization will be executed first and then the initialization switch set to false to preclude initializations from being executed on subsequent iterations of the main monitor loop. Since all submonitors return to the top of the main monitor loop, the top of the main monitor loop resets the stack pointer to eliminate any pending operations in the submonitor loop. This also allows error handling to be executed by jumping to the top of the main monitor.

Working	Test	Purpose
4000H	100H	jump to 4000h offset to protect loading rom
4000H	4000H	create stack and do any required hardware initializations, then jump to main monitor loop
		<pre> @spvsr main monitor loop reset stack pointer on switch 1 jump to submonitor 1 on switch 2 jump to submonitor 2 on switch 3 jump to submonitor 3 jump to @spvsr @submonitor 1 initialize variables User defined initializations switch 1 = false jump to @spvsr @submonitor 2 Call Procedures On Condition do call Task Call Procedures On Condition do call Task . . If switch 2 = false jump to @spvsr jump to @submonitor 2 @submonitor 3 Call Procedures On Condition do call Task Call Procedures On Condition do call Task . . If switch 3 = false jump to @spvsr jump to @submonitor 3 Procedures Tasks . . Variables Bottom of Stack Top of Stack </pre>
32767D	32767D	Monitor for Z-80 Realization

Figure 3

The library supports testing the monitor prior to building hardware and programming eproms by the use of two switches set in GLOBALS.DAT. These switches are the DEBUG

and NOROM switches. Three modes are allowed: Standard Board with loading rom, program in an ALTOS CP/M computer I/O through standard BDOS calls, and Standard Board without loading rom. The default mode is for both of the switches to be false. This allows loading of programs for test into the ram memory of the standard board system, using an ALTOS computer. The BDOS jump location is also defined in GLOBALS.DAT, to allow changes for computer with a BDOS jump location other than 5.

By setting the DEBUG switch to true to activate conditional assembly of test components, the conditional assembly relocates the beginning of the monitor to 100h. In the standard board system this area would normally have the loading monitor. That monitor has provisions for offsetting the interrupt locations to 4000h. To keep the rest of the program in the same relative position during the debug mode, a jump instruction is put at 100h to jump to 4000h. This will cause all code to be identical above 4000h to the breadboard system. This allows the use of a CP/M-based machine to test the software prior to hardware construction. The top of ram, where the stack would normally be located, is moved down to the top of the temporary program area in CP/M, to location 32767. Inputs that would normally be an I/O request to a input board will be transformed into BDOS calls for input at the keyboard. Outputs will be handled as a BDOS request for display to the console. In both cases the BDOS call includes a description of the board the input

or output would be going to as well as the value of the respective input or output.

In contrast to the test monitor the working monitor will have its initialization assembly begin at 4000h. This is the normal starting point for the NPS Prolog loading rom. This location was chosen to allow ease in starting the monitor program.

If a rom is actually burned and the nps loading rom is removed, then the conditional assembly must start at 66h. If a non-maskable interrupt, (generated by a reset button), is issued to the system, it will begin executing code at 66h. This permits the user of the system to use the reset button as a start button as well as a trouble button. The ROM is banked to start at 0000h, the locations 0000 to 0065h, normally unused. These locations will have no operation codes placed there. Any maskable interrupt can then be accidentally triggered without crashing the system. The maskable interrupt would merely jump to 38h, and do NOP's until it reached the initialization routine for the monitor.

B. ARITHMETIC

The format for the single byte integer operations used by the 8080 realizations has been retained in the Z80 realization. It is a two's complement arithmetic. The add and subtraction are essentially the same as in the Intel

library. The multiplication and division have been changed, since negative numbers were not correctly implemented. In addition, the option of checking the single byte operations has been added by checking for an overflow after performing the indicated operation. If an overflow occurs, the result will be set to the largest representable positive or negative number based on the sign of the result.

The format for the double byte integer operations is the same as the Intel 8080. The format is stored with the least significant byte first in memory. By using this format, the double byte operations in the Z80 instruction set can be used directly. This particular hardware instruction is used because it takes less time to fetch two bytes with one instruction, than fetching a single byte at a time and using two instructions. Once again the addition and subtraction are the same as the Intel 8080, since the same machine instruction is used for the operation. The multiplication and the division have been changed to treat negative numbers correctly. The check for overflow is done in a similar manner as the check done on the single precision arithmetic.

The format for the floating point operations is a departure from those done in the Intel 8080 library, since a floating point chip is not used in the Z80 library. The original approach was to do floating point using the IEEE standard single precision format. Further study of this approach has shown it to be impractical for two reasons. The first reason is the Z80 is a two's complement machine

and the IEEE format requires sign magnitude operations. The second reason is the packing of the IEEE format across byte boundaries. To use the IEEE format would require converting the sign magnitude number to two's complement and shifting the number to a usable format. This would have to be done before and after every arithmetic operation. To eliminate the overhead of the transformation, a primitive has been made to convert the format used in the realization to and from the IEEE format. The cost to implement the change has been the use of an additional byte of storage. The format used for the Z80 floating point is:

Exponent sign mantissa

39.....32 31 30.....0

The exponent is an eight bit number as in the IEEE standard, however it does not have an offset. Instead it is represented as a two's complement number. The mantissa is represented as a four byte two's complement number. The mantissa is fetched and stored two bytes at a time, so the position of bytes 1, 2, 3, 4 are stored as 2, 1, 4, 3. The additional byte is necessary to preserve the accuracy for rounding to the IEEE format. The leading 1 is expressed, rather than being encoded as in the IEEE format.

C. CONTROL STRUCTURES

The control structures can be divided into two categories, selection in straight line code and subroutines. The distinction is made because of the manipulation that

Ross's timing analyzer does with the order that conditions are tested and tasks selected. In the first category four constructs are supported: IF, While, JUMP-ON-TRUE, and JUMP-ON-FALSE. The IF and the WHILE support the CSDL language constructs directly, that is, there are primitives in two parts that will directly perform either of the two selections. They are used by placing the first portion of the primitive before the conditionally executed code and the second part of the primitive directly following the code. In the case of the IF, if the condition is not true, then a jump is done to the code immediately following the second portion of the if primitive. The JUMP-ON-TRUE, and JUMP-ON-FALSE support the CSDE compiler with more primitive operations for compilation into higher constructs. In all of these primitives, selection around straight line code is involved, that is, these primitives will cause a section of code to be included based on a condition, but will not directly support a construct such as an "else".

The second general category contains the control structures effecting subroutines. They are TABENT, PROC and two versions of EXITPROC. The purpose of these control structures is to allow the timing analyzer to manipulate the order that conditions/tasks are called to guarantee the maximum timing. To manipulate the timing the timing analyzer will change the order in which condition/tasks are polled as well as duplicating some condition/tasks to insure that they are polled often enough to ensure the timing

guarantee. This manipulation is done by making successive entries using the primitive TABENT, which will cause a entry of a condition/task in the polling loop. The primitive TABENT uses subroutines in two ways. First it uses an unconditional call to a subroutine to evaluate a condition. After returning from the conditional evaluation, a second subroutine call is made to the task based on the results of the condition. This is a change from the method that Ross used in constructing his polling loop. He chose to minimize the amount of ram required at the expense of increasing the number of jumps required to execute a contingency/task pair. The primitive combines the functions of Ross's primitives TABENT and TABACCP2 into a single primitive. The increase in memory will only become a problem when there are great differences in the timing requirement of different contingency/tasks. The great difference will cause the timing analyzer to make multiple entries of the contingency/task with the shortest time constraint. PROC marks the beginning of a subroutine. EXITPROC provides both a conditional and an unconditional return.

D. INTERRUPTS

The use of interrupts was not implemented in the 8080 library. Interrupts can offer the advantage of faster response to a particular contingency, but they add additional execution time to the contingency/task pairs in the form of interrupt overhead. Since the very nature of

the design system is guaranteed maximum timing, the faster response to a particular task is not a major issue as long as the overall timing constraints are met.

The non-maskable interrupt is used by the Z-80 library as a means to start the controller. This is implemented by starting the definition of the variables, input output/ports, hardware initializations and stack initialization at 66h, the non-maskable interrupt location in the Z-80. From here the execution will take the program into a continuous polling loop of the contingency task pairs.

A problem arises in trying to control tasks where speed is computed by the design system. Currently, the design system guarantees that a task will be tested within a certain time constraint, but it can be tested earlier. This can occur when all the conditions are false and no tasks are performed. In this case the only time required to execute the polling loop is the time to do the actual test of conditions. Within the structure of the primitives there is no way to make them run in a fixed amount of time, since the arithmetic operations take a different amount of time depending on the particular input values. Also the polling loop's overall timing changes depends on how many of the tasks are executed. The only guarantee the system gives is that the time the monitor will take to execute all the tasks and tests of conditions will be less than some maximum. An external reference is needed to insure a fixed amount of

time has elapsed. In the case of the 8080 library a clock was added to the system to give an external reference. To minimize the cost and keep the system simple the Z-80 library uses the CTC chip. The particular standard board used contains a counter timer chip with three channels as well as the Z80A and provisions for adding on-board memory. One of the channels in the CTC is used to generate a maskable interrupt in a fixed period of time. This interrupt will be handled by incrementing a word in memory that represents the current time. The details of the implementation of the clock interrupt primitive are in appendix C. Brief conditions will be recorded by using another channel on the CTC to count the occurrences independent of the operation of the cpu. This is implemented through another primitive that connects the external signal directly to the third channel of the CTC chip. A second primitive is used to read the counter in the channel and reset the counter to zero.

This particular clock primitive has an interrupt rate of a millisecond. It is included for use, but has a major disadvantage in overhead. The interrupt service routine requires 100 clock cycles to implement. This equates to 2.5% of the available monitor time. The interrupt rate can be reduced to every 10 milliseconds for an overhead of .25%. This in turn makes the polling loop longer to accommodate the accuracy of the clock. Because the exact point of interrupt cannot be determined prior to execution, all of



the times computed for the contingency/tasks must have the interrupt service time of the clock added to their maximum time. Several executions of the interrupt are required to built up some accuracy in the computation of the stored time in 10's of milliseconds. This would be adequate for events that are very slow (in the realm of seconds), but inadequate for those taking fractions of a second.

E. INPUT/OUTPUT DEVICES

The selection of input or output software primitives has an additional effect of adding an associated hardware primitive. The hardware primitive in turn require some initialization prior to operation. The hardware initialization is done before any user defined initialization in order to hide the details from the user. Since the user is not required to consciously select all the I/O references in the very beginning of his program, an I/O board can be added with any I/O reference. A method to add the required initialization is needed that could be placed anywhere without effecting the runtime execution of primitives. The compilations of the hardware and software is currently done in a single pass. This also requires that the actual initialization code be added to the output code file at the time of the request. A linked list was used to allow the random placement of any required hardware initializations. This does have an undesirable effect on timing. To keep other segments of code from executing the

initialization, a unconditional jump is used before the initialization. Since it isn't known when this is executed by the monitor as part of a condition, the hardware adds the time for an unconditional jump anytime an initialization is required.

1. Onboard Cpu Three Channel Counter Timer Chip

The solution to the interrupt overhead dilemma was to eliminate the interrupt driven clock. The second clock primitive uses two of the CTC chip's channels. One is used as in the interrupt case to create a one millisecond clock. The second is used in lieu of a memory location and an interrupt service routine. The second channel's clock input is tied to the output of the one millisecond channel. The capacity of the channel is 65,536 milliseconds or just about a minute. This short time period is a problem for some applications that may require more than a minute of running time, so a 25 millisecond clock primitive was added. The construction was the same as the 1 millisecond clock except for the variable loaded into the channel 0 counter. The other service primitives remain the same, except that the magnitude returned represents a different elapsed time. There are two additional primitives related to this clock: one to read the time and one to reset the clock to 0. This leaves one channel in the CTC on the cpu board for other purposes.

The remaining channel is used to count fleeting external events that potentially take fractions of a monitor loop execution. That channel is initialized and read/reset via another primitive. Since the CTC chip was included with the cpu board and contains only 3 channels, the primitives will allow a single counting channel and a clock or three counting channels. No provisions were made in the library to add additional counter timer chips nor to multiplex the channel over several inputs.

2. 8 Bit Analog To Digital Conversion Board

Input of analog voltages was done by an analog-to-digital primitive using a MOSTEK MDX-A/D8 board. The accuracy of this board is limited to 8 bits. The primitive will allow up to 32 analog signals, using two separate hardware boards. Each of the requested signals are counted using a global variable NATODE. This cannot be used directly since each board has a single port address. To decode the address a second variable NATODP is used to indicate the actual address used. Up to 16 signals can be multiplexed to that port. For signals less than a board's capacity, the NATODE variable was used to compute the proper pin for the input signal. For signals greater than one board, 16 was subtracted from the NATODE and that number was used to assign the input signal to the second board. To keep from changing the actual value of NATODE, a scratch variable is changed and that value is printed in place of NATODE. The hardware configurations of the board are

controlled by the hardware primitive if a board is required, however, the analog signal assignment to that board was done by the software primitive. This is in contrast to the other primitives where all the signals were assigned pin by only the hardware primitive. This was done in the software listing, because of the multiple signals assigned to a single board.

3. 8 Bit Digital To Analog Conversion Board

Output of analog signals is planned through a digital to analog standard bus board. The configuration of the board is analogous to that of the analog to digital board. The signal capacity is also planned to be the same.

4. 64 Port 8 Bit Standard Bus To Digital I/O Bus

This board was intended to be used to turn devices on and off. However, the MOSTEK MDX-DIOB1 board was found to be merely an interface from the Standard Bus to the Digital I/O bus. The board was designed to multiplex 64 channel through a single standard board card. The board has the capability to send or receive 8 bit wide signals to a MOSTEK digital I/O board. This is the capability that was desired when the board was purchased. However, the digital I/O board that actually controlled external inputs was not purchased. To retain some of its intended capability, turning leds on, the address lines were wired to turn on lamps when a particular addresses was outputted. This is just an intermediate primitive to make use of the board until the digital I/O board can be purchased. Because the



light is only illuminated when the address is strobed, this is not an adequate primitive for a working system. To keep the light illuminated long enough for a person to see the lamp, the address is strobed for several seconds, making the execution time for this primitive long.

5. Keyboard/Display Card

This board was added to the realization library to provide limited front panel access to the controller. This particular board has a programmable key pad from input and light emitting diodes and segment alphanumeric output capability. Because of the many functions on the card, each function is provided as a separate primitive. The card is include only once and is initialized in the hardware primitive. The two rocker switches are tested using primitive S.ROCKER. The board also contains eight light emitting diodes that are controlled using primitive S.OUTLED. In the case of S.OUTLED the individual LED is turned on or off based on a boolean value. The Keyboard can be defined using S.INKEY. Pressing the key associated with a boolean flag causes the flag to be complemented. All three of these primitives assign the key or light in the software primitive. None of the primitives associated with the keyboard/display card will add any additional boards, but will issue an error message if more lights or keys are requested than are available on a single board. The last primitive associate with this card is S.OUTDIGIT. This primitive will print a message to the alphanumeric digits on

the card in the form of a scrolling banner. The code for this primitive was provided with the board. It has a distinct penalty in that it uses 500 bytes.

6. Dual UART Board

This board was provided along with a bootstrap rom on the cpu board as a method to load programs into the controller. This provided the means of testing the boards program using a ram board. The final controller does not require this board unless specifically determined by the application. No primitive is required to use this board, but its installation is required if the NPS Bootstrap loading prom is used. That prom assumes that the terminal is connected to the A port and that another Altos computer is connected to the B port of the board. In final use the program would be contained in a rom or a prom, making the loading unnecessary.

F. TESTING

Testing of the primitives has been done in three phases. First, the primitives have been functionally tested individually in the course of writing them. They have all passed an assembly and have been executed individually using a debugger to insure that the outputs are appropriate. The code associate with an initialization of a board has been adapted from the manufacturers examples and has been assembled. It has not been run on the Prolog System to

check its accuracy. The primitives have been tested individually, but have not been tested for interactions.

Because the controllers written with these primitives can be potentially quite complex, two additional methods of testing the controller's program are provided. The first is the debug switch. This is a global variable that allows a conditional assembly of the primitives. The purpose of the switch is to allow the program to be run on a micro computer that has a z-80 processor and the CPM operating system. The switch causes all outputs to be sent to the console and all inputs to be requested from the keyboard via a message to the console. To implement these features the additional primitives wrtbin and messout are included. Their purposes are binary output and message output respectively. The second method is the use of the serial input/output board and bootstrap rom to load the program from a microcomputer into the controller. This permits the controller to be run and controlled from the microcomputer. This also permits the controller to be tested with its I/O boards for proper operation prior to loading the program into a prom.

G. FORMATTING OF PRIMITIVES

The format of the primitives is driven by several factors. The most obvious is the instruction set of the cpu, and the next is the form of the arguments. The form of the arguments used in the Z-80 library were typically rslt, arg1, or rslt, arg1, arg2. This is the same form as the

Intel 8080 library. In the construction of the primitives all results are stored at the conclusion of a primitive. In the case of a complex evaluation this might cause a value to be stored in memory only to be fetched by the next primitive. This is very costly in terms of execution time. During the construction of the primitives care was taken to insure that all results are in the registers that initially contain argument2. This could allow an auxiliary set of primitives to be developed in the future to take advantage of the chaining of arithmetic operations. The format of the title line and the reserve words are defined in Ross's thesis and have not changed. [Ref. 19: pp. 79-85]

Since the primitives were tested individually on a CPM machine prior to placing them in the library, several stages of collating and compromises in editing speed were made to insure integrity of the primitive. All the code and text associated with the primitive was kept in a single file by primitive. To debug the primitive a header file and trailer file was added to allow the proper assembly of each single primitive. These three files were combined into the actual test file that was assembled and debugged. All the text that was not part of the code for the primitive but was necessary for the operation of the primitive in the design system was commented out in column one. Data on the execution time of each line of code was kept as a comment after the code. It is in the form of ;?m ??t ?b comment. The "m" was the memory cycle time of the instruction. The

"t" was the machine cycle time of the instruction and the "b" was the number of ROM bytes used by the instruction.

The code format used by the design system and the z80 assembler can cause conflicts. In order to insure the integrity of a primitive, the entire primitive was stored in a single file. All the text outside of the markers begin stext and endtext is used by the design system and is not compatible with the assembler. To keep this text from causing assembly errors, this text was commented out using the ";". The primitives when written did not have blanks in the first five columns. Ross's program NEWCSDL requires that the library have a line number on each line. This can be done by the program FIXIT however the program does not append the program line to a line number, but rather changes the first five columns to an appropriate line index. The primitive had to be reformatted by adding five blanks to the beginning of the line before sending it to the VAX. They were run through a program that removed the ";" if it existed in the first column and stuffed five spaces in front of all lines. The primitives were then appended together and sent by modem to the VAX. The VAX uses a carriage return to indicate the end of a line. When sending a file the terminal program sends a carriage return and a line feed, giving each line an extra linefeed. After receipt of the file, all the line feeds added by the terminal program had to be edited out. At this point the file is in a format acceptable by FIXIT.

1. Pollock Fixit

Pollock expressed difficulty in trying to use Ross's primitive format and he wrote a FORTRAN program to correct the placement of pointers used in the title line [Ref. 20: pp. 23-24]. The problems originate in the structure of the primitive file as designed by Ross. Each library file is required to have an alphabetic index of all primitive title lines. Implicit in this is the requirement to number all the lines. Ross's format for the numbering was in the format "vxxxx", where x represents the number of the line. Within each title line there are four additional pointers, first INCL, first CALC, first line of primitive and last line of primitive. All of these pointers make any change extremely difficult. Even simple changes require running this program since the VAX editor creates a variable length record file and Ross's program requires a fixed length record file.

Pollock's program FIXIT was written to minimize the effect of the format requirement imposed by Ross's program. The program was written for a Cyber computer. The program was transported here via tape, however the program did not run due to differences in machines and program errors. The program is now corrected and working on a VAX11/780. Much of the program's problems were in the differences in I/O and word size in the two machines. The program is now set up to take a file of primitives named "inname.dat". It produces a file "outname.dat" of correctly formatted primitives that

includes a sorted title directory at the beginning of the file with pointers to the individual primitives. Without this utility, change to the realization would be very tedious.

2. Ross's Newcsdl

When running NEWCSDL there are options to produce a trace by subroutine of the program execution. This is a very necessary utility, since the program is very complex and sensitive to input format. The detailed trace produced by NEWCSDL is very necessary because of the size of the program. This became painfully obvious while trying to get FIXIT to give an acceptable input file for the primitives. The error messages produced by NEWCSDL were incorrect due to a format error. The carriage control has been corrected, and the messages are now intelligible. In using NEWCSDL trace option in full mode, every subroutine called is printed to a log file along with the movement of key data. The carriage control that was a problem in the output of messages was also the principal problem in the input of the primitives. The carriage control character kept moving the data of the input file over by one character, causing the data to be in the wrong column. The requirement for fixed column reads in this program precludes any editing of a primitive file without running it through FIXIT now named FORMAT. The name was changed to minimize the confusion between the various versions of the program. FORMAT, in addition to correcting any pointer inconsistencies also

converts the file from a variable format to a fixed 80 character record format, which is the required input format for NEWCSDL.

In writing the primitives, the desirability of a modulo operator in the calc function became evident. The addition of multiple boards makes this desirable in assigning pinouts on the additional boards. Currently the number of ports by type requested is accumulated as a variable in GLOBALS.DAT. Ross calcit subroutine was examined to determine if this was feasible. Currently, the only primitives that could include multiple boards are the 8 bit analog to digital primitive and the 8 bit digital to analog primitive. Because of this limited number of boards requiring a modulo function, a simple scratch variable and subtraction of any values greater than the number of ports on the board was used instead. If the number of replicated board grows, then it may be worthwhile in the future to change NEWCSDL to include this function.

Two additional files used by NEWCSDL are potentially affected by new libraries. The first file MONITOR.DAT contains primitives that are used all the time by NEWCSDL and are included regardless of the application. Because the intermediate table containing the order of the contingency tasks has been eliminated, one primitive TAACP2 has been eliminated and the code in NEWCSDL needs to be modified to eliminate the reference to that primitive. The second file, GLOBALS.DAT, contains the names of global



variables used by the primitive library. Because of the change in orientation of the library from individual components to boards, many globals were no longer needed. A list of the global variables used with this library and their application is given in appendix B.

3. Walden's Data Base Input

Walden designed a database system to eliminate the need for the various files used in NEWCSDL. No code was available for test at the time of the implementation of this thesis, so no attempt was made to try inputting any of the primitive library into a database. The method of input into the data base specified by Walden was to type all data in at a terminal. This is a reasonable method for the title lines and some of the smaller files. It does not appear to be a feasible method for entering a primitive library because of the size of the data. A data base eliminates the need to use the FORMAT program to correct the pointers and eliminates some of the overhead on CALC, and INCLUDE. However it would require typing all of the primitive's assembly code that has been previously debugged, with the attendant errors and duplication of work. To be really usable, the data base needs a method of getting the assembly code associated with a primitive from its file used for testing and debugging. By doing this the code could be transferred with a minimal amount of error. The code would still be available for separate assembly and debugging by the primitive's author. [Ref. 21]

V. RESULTS AND CONCLUSIONS

Preliminary results of Riley's thesis indicate that the controller can be built using the design system. [Ref. 22] This particular controller does not overly tax the design system or the primitive library, but does provide a functional test.

In the course of constructing the library, integration of the various tools became the major problem. In constructing any library an assembler and debugger are necessary. The Z-80 has a number of assembler programs, however, they do not run on a VAX11/780. The transport of the assembly file to the mainframe after construction can be tedious. My choice was using a 300 baud modem. This was not much of a problem when the primitives were few in number, but became a real logistics problem as the library neared completion.

The implementation of an interrupt driven controller has turned out not to be desirable because of the additional overhead. This was tried in the form of an interrupt driven clock for the primitive library. The clock was using about 2.5% of the monitor's available time. This required no selection of competing interrupts, since there was only the single interrupt in this case. The overhead was also the minimal possible, since only the AF and HL registers were

saved. If all the primitives had the potential for interrupt, then the AF, BC, DE, HL, IX, AF' BC' DE' and HL' would also have to be saved. For comparison, saving all the registers would have increased the overhead to 7.5%. This still would not include the time necessary to select which contingency/task pair should be executed. At this point the interrupt driven monitor was abandoned.

Further primitives can be added to the z-80 library to increase the number of arithmetic and control functions available, but this may be time poorly spent. In building a good design system a suite of libraries is necessary. The Z-80 is the low end of the library in terms of performance and cost. Because the costs of processors continue to fall, it may be desirable to include the most extensive of libraries only in higher performance chips.

APPENDIX A

PRIMITIVE TITLE INDEX

This appendix was created from the index of the primitive title lines. Where there are multiple primitives with the same name, but different precisions, only one name is listed. The function of the primitive is then briefly described, and its limitations.

NAME	FUNCTION
h.atod	Include a 8 bit analog to digital conversion board
h.cardcage	Include a 8 slot card cage and power supply in the primitive listing.
h.clock	Detail the connections of the counter timer chip on the cpu board to produce a 1 millisecond or a 25 millisecond clock out of channels 0 and 1.
h.dtoa	Include the hardware for a 8 bit digital to analog conversion board
h.inout	Include a 8 bit 64 channel standard bus to digital i/o board with wiring to illuminate up to eight lights using the address lines on the card
h.memory	Hardware primitive to include a 16k ram board based on total ram and rom requirements. This particular board is a battery backup board and can also be used as a quasi rom if a write inhibit switch is on. The board can be disabled in 4k segments, making that particular segment nonwriteable by the cpu.
h.processor	Hardware primitive to include a Zilog Z-80a with a 4mhz clock and a three channel counter timer chip on a single board. Included also is a bootstrap rom using the first 4k of address space.
h.tcardcage	This is a primitive that is invoked when a card slot is requested. It checks that the number of slots requested does not exceed the number available.
s.add	Comes in a variety of forms to support byte, two byte and floating point addition. It provides no error checking on the result of the

computation.

- s.addck** Comes in a variety of forms to support byte, two byte and floating point addition. It provides error checking and will not allow the result to have an inappropriate sign. If an overflow is made then, it will put the largest possible value in the result.
- s.and** Perform a logical and
- s.assign** Performs an assignment operation.
- s.assigncons** Assigns a constant value to a previously defined variable. This does not reserve space for the variable, only puts a specific value in the variable.
- s.atod** Primitive to perform a analog to digital conversion
- s.blockcons** Primitive to mark the beginning of a submonitor block
- s.blockend** Primitive to mark the end of a submonitor block
- s.blockexit** Primitive to leave a submonitor and reset all pending operations
- s.blockstart** Primitive to cause a submonitor to be executed
- s.clockcons** Primitive to create an interrupt driven clock
- s.clockcons** Primitive to create a clock using counter timer chip channels 0 and 1. No interrupt is involved. The accuracy of the time tick is once every millisecond. The time is accumulated in channel 1 as a 16 bit down counter. The time can be read by using **s.rdtype**. This primitive will latch the current time to an output buffer and subsequently input the latched time.
- s.clockcon25** Primitive to create a clock using counter timer chip channels 0 and 1. No interrupt is involved. The accuracy of the time tick is once every 25 milliseconds. The time is accumulated in channel 1 as a 16 bit down counter. The time can be read by using **s.rdtype**. This primitive will latch the current time to an output buffer and subsequently input the latched time.
- s.cold** Primitive cause the system to do a cold boot.

That is reinitialize all hardware and perform any user directed initializations.

s.cons	Define a constant. That is put a value in the rom position of memory. Comes in version for byte, two byte and floating point constants
s.div	Comes in version for byte, two byte and floating point divisions.
s.dtoa	Software primitive to perform an 8 bit digital to analog conversion.
s.end	Primitive that must be last. Indicated the end of program and includes some necessary pointers for hardware initialization.
s.eq	Performs comparison of byte or word values for the condition of equality and outputs boolean result
s.every	Forces execution of monitor every time by making condition always true.
s.exitproc	Marks the end of a procedure that is executed as a subroutine. When used with a conditional call, this primitive will first set a boolean variable with the same name as the procedure to false prior to executing a return. If it is used in conjunction with a unconditional procedure then it will simple execute a return.
s.float	Converts a two byte variable to a floating point variable.
s.forend	Marks end of a FOR construction
s.forstart	Creates a FOR variable= date from lower to upper.
s.fptoieee	Converts floating point format in controller to IEEE single precision format.
s.ge	Performs comparison of byte or word values for the condition of greater than or equal and outputs boolean result
s.gt	Performs comparison of byte or word values for the condition of greater than and outputs boolean result
s.ifcons	Marks to to an if construction

s.ifend	Marks the end of an if construction
s.initalcons	Marks the beginning of user defined initialization requirements
s.initalend	Marks the end of user defined initialization requirements. Implied in the end is setting the flag associated with the user defined initialization to false and jumping to the top of the main monitor loop.
s.jmpf	Causes a jump to a location if a variable is false
s.jmpt	Causes a jump to a location if a variable is true
s.le	Performs comparison of byte or word values for the condition of less than or equal and outputs boolean result
s.loc	Marks a portion of a program with a label.
s.lt	Performs comparison of byte or word values for the condition of less than and outputs boolean result
s.main	This construction is always required. It initializes some basic pointers and creates a stack.
s.messout	Sends a message to the output device. It is used only in the debug mode.
s.monitor	Creates the top of the main polling loop and reinitializes the stack pointer
s.mult	Comes in a variety of forms to support byte, two byte and floating point multiplication. It provides no error checking on the result of the computation.
s.ne	Performs comparison of byte or word values for the condition of not equal and outputs boolean result
s.not	Performs a logical not.
s.or	Performs a logical or
s.out	Output a signal to a port.
s.perform	Primitive to invoke a procedure subroutine call.

s.proc	Primitive to mark the beginning of a conditional or unconditional subroutine call. It is used in conjunction with s.exitproc to mark the end of the subroutine.
s.rdtype	Reads the time from the CTC chip on the cpu board. Time is represented in either milliseconds or 25 milliseconds depending on which clock construction has been used. In both cases the actual number read is a two byte number. The clock is a down counter, so that the elapse time goes from a negative number to a positive number to zero and then repeats starting with a negative number.
s.start	Forces monitor execution provided for compatibility with older library. Not needed since monitor starts at nonmaskable interrupt location.
s.sub	Comes in a variety of forms to support byte, two byte and floating point subtraction. It provides no error checking on the result of the computation.
s.subck	Comes in a variety of forms to support byte, two byte and floating point subtraction. It provides error checking and will not allow the result to have an inappropriate sign. If an overflow is made then, it will put the largest possible value in the result.
s.tabend	Marks the end of the main monitor polling loop.
s.tabent	Put an entry in a polling loop. It is used with both the main and submonitor loops.
s.var	Defines a variable in ram
s.warm	Performs a warm boot. That is jump to the top of the main polling loop, reinitialize the stack, and perform any user defined initializations.
s.whend	Mark the end of a while construction
s.whilecon	Mark the beginning of a while construction. Performs the test of a condition, if false jumps to the next instruction past the location marked by whend.
s.wrtbin	Used in the debug mode to output a location to

the screen in binary.

s.xor Perform an exclusive or.

s.tabaccp2 Is a dummy primitive to provide compatibility with the 8080 realization listing. In the 8080 listing tabent is divided into two primitives tabent and tabaccp2. This requires less memory if there are extremes in the time requirements of different contingencies. By combining the two primitives an intermediate table is eliminated and the execution time is increased by eliminating two unconditional jumps.

APPENDIX B

GLOBAL VARIABLE LISTING

This appendix lists exactly the file GLOBALS.DAT under the column name. Included in the listing are the initial values of the variable. Since the variables are limited to six characters, the purpose and limitations are not readily apparent. The purpose of the variable and its use are described in the next column.

NAME	PURPOSE
arnd 0.	Is a pointer in the form of "@@<arnd>". Its purpose is to allow sections of code to be placed anywhere. This allows initializations to be placed with a primitive. In this way the primitive can be executed and the initialization is jumped around. The only way to access the initialization is to use the inlnk chain.
chips 0.	Is a count of the number of integrated circuits that have been added.
debug 0.	Is a flag that will cause the primitives to be conditionally assembled to produce input and output through standard BDOS calls. The default is to use the normal input and output boards. To use a cpm microcomputer to test the primitive, debug must be set to 1. This can be done as an exit to the globals.dat file.
initlk0.	Is a variable that is used in a pointer chain for the initialization of hardware. The actual form is @i<initlk>. This for the first link would appear as @i0, the second @i1.
keybrd0.	Is a boolean variable indicating if the 7303 keyboard/display board has been requested by any primitive. This primitive was necessary to eliminate the potential of several copies of the board being requested by different primitives. There are several primitives because of the number of separate functions that are included on the board.
natode0.	Is a variable enumerating the number of analog to digital ports that have been requested. The hardware primitive will

indicate a failure if the number is greater than 32.

- natodp0. Is a variable containing the address of the I/O port. It is initially 0. If more than 16 atod ports are requested then the address is changed to 4.
- ndtoae0. Is a variable enumerating the number of digital to analog ports that have been requested. The hardware primitive will indicate a failure if the number is greater than 32.
- ndtoap0. Is a variable containing the address of the I/O port. Currently, not dtoa boards are available for the system, so the exact address has not been determined.
- ninout0. Is the number of inout ports that have been request for the mostek mdx-diobl board. An error is produced it the number requested exceeds 64.
- nkey 0. Is a variable indicating the number of push button keys that have been requested.
- nled -1. Is the number of light emitting diodes that have been requested. The maximum is eight. An error will be generated if more than eight are requested. The lights are number zero thru seven.
- nodgt 0. Is the number of digits of the alphanumeric display that are requested.
- norom 0. Is a flag indicating the absence of the loading rom. If the controller is to be used without external support, this flag must be set to 1. This will cause the conditional assembly to start at 66d and will allow the normal use of interrupts. If the controller is being simulated on a cpm computer, this flag should be 0. The normal assembly has a jump to 4000h at 100h to accommodate the use of cpm. This does not effect the use of the loading rom, since the location 100h can't be altered by the loader.
- nrockr0. Is the number of rocker switches that have been requested for the 7303 keyboard/display board. Currently this is limited to two switches. Requesting more than two will

cause a error.

- ramptr0. Keeps a count on the number of bytes of storage that have been requested in a RAM area. This counter starts at zero, but will be moved to the top of the address space and decremented as memory is requested for variables and stack space.
- romptr0. Keeps a count on the number of bytes of storage that have been requested in a ROM area. This counter starts at 0 and is incremented as instructions are added.
- scrтч0. Is a scratch variable. It is used in various was by different primitives to compute intermediate results.
- slot 0. Indicate the slot in the card cage that a board will use. An error will be generated if more than eight cards are used.

APPENDIX C

ZILOG-80 REALIZATION LISTING

This appendix contains the listing of the library as it is required to be formatted. The first line of the library contains the title of the library along with the cpu's clock period and memory speed in quarter mircoseconds. This is followed by the alphabetical index of all the title lines. The individual primitives are listed after the index. The format of the individual primitive repeats the title line, has comments describing the primitive, then actual code and is unbounded until another title lines is encountered. This makes it possible to have multiple segments of code interspersed with calc statements. Not shown because it is an extra line, is the requirement to a line after the last line of the last primitive. If this is not included, the last line of the last primitive will not be included, since the FORMAT program is looking for a last dummy primitive to mark the end of the library.

The library is listed vertically to allow the fully 80 columns to be displayed.


```

v0000 z80 cpu
v2024h.atod (: : 552,30,15,7,8,2024,2064)
v2090h.cardcage (: : 0,0,2090,2094)
v2238h.clock (: : 0,0,2238,2243)
v2123h.dtoa (: : 552,30,15,4,5,2123,2137)
v1999h.inout (: : 552,30,15,5,6,1999,2023)
v2244h.keydisplay (: : 6,8,2244,2263)
v2095h.memory (: : 2,3,2095,2115)
v2065h.processor (: : 2,3,2065,2089)
v2116h.tcadcage (: : 0,0,2116,2122)
v2264h.uart (: : 0,0,2264,2278)
v0186s.add (rslt,arg1,arg2:0,16,0,16,0,16,0,16:13,71,21,9,0,186,195)
v0526s.add (rslt,arg1,arg2:0,8,0,8,0,8,0,8:10,43,13,9,0,526,535)
v0501s.adduck (rslt,arg1,arg2:0,8,0,8,0,8,0,8:23,80,24,14,0,501,515)
v0799s.adduck (rslt,arg1,arg2:0,16,0,16,0,16,0,16:33,144,39,18,0,799,817)
v0175s.and (rslt,arg1,arg2:0,8,0,8,0,8,0,8:11,47,14,10,0,175,185)
v0826s.assign (var,data:0,8,0,8,6,26,8,7,0,826,833)
v0834s.assign (var,data:0,16,0,16,7,36,11,7,0,834,841)
v0818s.assigncons (var,data:0,8,0,8,5,20,6,7,0,818,825)
v1520s.atod (signam:0,8:43,641,23,9,11,1520,1610)
v1620s.blockcons (submon:0,255:1,4,1,9,0,1620,1629)
v1733s.blockend (switch,submon:0,8,0,255:10,37,10,9,0,1733,1742)
v1611s.blockexit (switch:0,8:5,20,5,8,0,1611,1619)
v1630s.blockstart (switch,submon:0,8,0,255:7,27,7,9,0,1630,1639)
v0669s.clockcons25 (: : 27,14,4,17,12,669,704)
v0139s.clockcons (: : 27,14,4,17,12,139,174)
v1826s.clockcons (: : 43,14,4,7,0,1826,1870)
v1760s.cold (: : 3,10,3,7,0,1760,1767)
v0850s.cons (nam,val, : 0,8:1,0,0,6,0,850,856)
v1070s.cons (nam,val, : 0,16:2,0,0,6,0,1070,1076)
v0790s.consfp (nam,val, : 0,32:5,0,0,8,0,790,798)
v0196s.conv816 (rslt,arg1:0,16,0,8:17,68,20,11,0,196,207)
v2138s.dlv (rslt,arg1,arg2:0,8,0,8,0,8,0,8:56,504,129,41,0,2138,2179)
v2180s.dlv (rslt,arg1,arg2:0,16,0,16,0,16,0,16:78,1458,374,57,0,2180,2237)
v1768s.dtoa (signam:0,8:43,641,23,10,12,1768,1825)
v1161s.end (: : 3,10,3,8,10,1161,1171)
v0341s.eq (rslt,arg1,arg2:0,8,0,8,0,8,0,8:15,72,20,13,0,341,354)
v0602s.eq (rslt,arg1,arg2:0,8,0,16,0,16,0,16:16,89,25,12,0,602,614)
v1172s.every (nam,: : 7,34,9,10,0,1172,1182)
v1058s.exitproc (nam,cont : : 6,30,8,11,0,1058,1069)
v0916s.fadd (rslt,arg1,arg2:0,32,0,32,0,32,0,32:205,2263,597,126,0,916,1042)
v2028s.float (rslt,arg1, : 0,32,0,16:46,528,143,59,0,208,267)
v1357s.fmul (rslt,arg1,arg2:0,32,0,32,0,32,0,32:204,2263,597,0,0,1357,1519)
v0462s.forend (indx,slab,elab:0,8,0,255:7,27,8,9,0,462,471)
v0563s.forstart (indx,lwr,upr,slab,elab:0,8,0,8,0,8,255:13,50,15,11,0,563,574)
v1086s.ftoleee (rslt,arg1:0,32,0,32, : 105,558,140,67,0,1086,1153)
v1183s.fsub (rslt,arg1,arg2:0,32,0,32,0,32,0,32:206,2263,597,126,0,1183,1309)
v0410s.ge (rslt,arg1,arg2:0,8,0,8,0,8,0,8:42,108,31,28,0,410,438)
v0575s.ge (rslt,arg1,arg2:0,8,0,16,0,16,0,16:45,118,34,26,0,575,601)
v0109s.gt (rslt,arg1,arg2:0,8,0,8,0,8,0,8:45,118,34,29,0,109,138)
v0642s.gt (rslt,arg1,arg2:0,8,0,16,0,16,0,16:45,118,34,26,0,642,668)
v1750s.ifcons (arg1,ifmark:0,8,0,255:7,27,7,9,0,1750,1759)
v1871s.ifend (ifmark:0,255:1,4,1,7,0,1871,1878)

```



```

v1879s.italcons(:,1,4,1,5,0,1879,1884)
v1885s.italend (:,8,30,8,9,0,1885,1894)
v0765s.inkey (arg1:0,8:11,47,14,11,0,765,776)
v1319s.jmpf (val,loc:0,8: 8,30,10,8,0,1319,1327)
v1310s.jmpt (val,loc:0,8:8,30,8,0,1310,1318)
v0472s.le (rs1t,arg1,arg2:0,8,0,8,0,42,108,31,28,0,472,500)
v0615s.le (rs1t,arg1,arg2:0,8,0,16,0,16:45,118,34,26,0,615,641)
v1154s.loc (loc:1,4,1,6,0,1154,1160)
v0366s.lt (rs1t,arg1,arg2:0,8,0,8,0,8:45,118,34,29,0,366,395)
v0536s.lt (rs1t,arg1,arg2:0,8,0,16,0,16:46,131,38,26,0,536,562)
v0705s.main (:,12,44,12,5,3,705,764)
v1903s.messout (arg1:0,16:...,0,0,1903,1917)
v1328s.monitor (:,7,27,7,10,0,1328,1338)
v1640s.mult (rs1t,arg1,arg2:0,16,0,16,0,16:39,1105,289,22,0,1640,1662)
v1688s.mult (rs1t,arg1,arg2:0,8,0,8,0,16:33,521,137,21,0,1688,1709)
v1710s.mult (rs1t,arg1,arg2:0,8,0,8,0,8:34,522,137,22,0,1710,1732)
v0396s.ne (rs1t,arg1,arg2:0,8,0,8,0,8:15,72,20,13,0,396,409)
v1979s.ne (rs1t,arg1,arg2:0,8,0,16,0,16:16,89,25,12,0,1979,1991)
v0332s.not (rs1t,arg1:0,8,0,8,7,30,9,8,0,332,340)
v0098s.or (rs1t,arg1,arg2:0,8,0,8,0,8:11,47,14,10,0,98,108)
v1663s.out (signam,direct:0,8:14,45,12,6,13,1663,1687)
v0777s.outdigit (arg1:0,8:11,47,14,0,0,777,789)
v0296s.outled (arg1:0,8:24,100,25,10,8,296,331)
v0889s.perform (name: : 3,17,5,6,0,889,895)
v0842s.proc (name:1,4,1,7,0,842,849)
v0279s.rdtme (rs1t:0,16:10,58,16,16,0,279,295)
v0439s.rocker (arg1:0,8:13,53,14,7,6,439,461)
v1992s.start (:,3,10,3,6,0,1992,1998)
v0355s.sub (rs1t,arg1,arg2:0,16,0,16,0,16:15,79,23,10,0,355,365)
v0516s.sub (rs1t,arg1,arg2:0,8,0,8,0,8:10,43,13,9,0,516,525)
v0896s.subck (rs1t,arg1,arg2:0,16,0,16,0,16:33,134,39,19,0,896,915)
v1043s.subck (rs1t,arg1,arg2:0,8,0,8,0,8:23,87,26,14,0,1043,1057)
v0868s.tabaccp2 (:...,0,0,868,879)
v1339s.tabend (:,3,10,3,6,0,1339,1345)
v0857s.tabent (fnc,task:10,51,15,10,0,857,867)
v0880s.var (name:0,8:0,0,0,3,0,880,888)
v1077s.var (name:0,16:0,0,0,3,0,1077,1085)
v1346s.varfp (name:0,16:0,0,0,3,0,1346,1356)
v1743s.warm (:,3,10,3,6,0,1743,1749)
v1895s.whend (whend,whntop:0,255:3,10,3,7,0,1895,1902)
v1918s.whilecon (arg1,whend,whntop:0,8,0,255:7,27,7,9,0,1918,1927)
v1928s.wrtbin (:...,0,0,1928,1978)
v0268s.xor (rs1t,arg1,arg2:0,8,0,8,0,8:11,47,14,10,0,268,278)
v0097 .end index
v0098s.or (rs1t,arg1,arg2:0,8,0,8,0,8:11,47,14,10,0,98,108)
v0099com primitive to perform logical or
v0100com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addr
v0101begin stext
v01021d a,<arg1> ;4m 13t 3b rs1t = arg1 .or. arg2
v01031d b, a ;1m 4t 1b
v01041d a,<arg2> ;4m 13t 3b
v0105or b ;1m 4t 1b
v01061d (<rs1t>),a ;4m 13t 3b

```



```

v0107endtext
v0108calc romptr=romptr+11
v0109s.gt      (rs1t,arg1,arg2:0,8,0,8,0,8,45,118,34,29,0,109,138)
v0110com primitive to perform comparison between 2 8-bit numbers
v0111com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0112begin text
v0113ld a,(<arg2>) ;4m 13t 3b if arg2 lt arg1 then rs1t=ffh
v0114ld b, a ;1m 4t 1b b=arg2
v0115ld a,(<arg1>) ;4m 13t 3b
v0116ld c, a ;1m 4t 1b c=arg1
v0117and a ;1m 4t 1b set sign flag of arg1
v0118jp p,$+00dh ;3m 10t 3b jump if arg1 is positive
v0119ld a, b ;1m 4t 1b arg1 = -
v0120and a ;1m 4t 1b set sign flag of arg2
v0121ld b, c ;1m 4t 1b arg2 .swap. arg1
v0122jp m, $+011h ;3m 10t 3b arg2 = - arg1 = - comp backwards
v0123ld a, 0 ;2m 7t 2b arg2 = + arg1 = - false
v0124jr $+013h ;3m 12t 2b
v0125ld a, b ;1m 4t 1b
v0126and a ;1m 4t 1b set sign flag of arg2
v0127ld a, c ;1m 4t 1b restore arg1 to accumulator
v0128jp p, $+007h ;3m 10t 3b arg2 = + arg1 = +
v0129ld a,11111111b;2m 7t 2b arg2 = - arg1 = + true
v0130jr $+009h ;3m 12t 2b
v0131cp b ;1m 4t 1b
v0132ld a,00000000b;2m 7t 2b result false arg2 >= arg1
v0133jp z, $+7 ;3m 10t 3b
v0134jp m, $+4 ;3m 10t 3b result true arg2 lt arg1
v0135cpl ;1m 4t 1b
v0136ld (<rs1t>),a ;4m 13t 3b
v0137endtext
v0138calc romptr=romptr+19
v0139s.clockcons (::27,14,4,17,12,139,174)
v0140com primitive to generate a 1 millisecond clock. the value of
v0141com current time is stored in the channel 1 counter of the cpu
v0142com ctc chip. to access this particular time rdtime must be used
v0143com to read/rest the time use rdrsttime primitive.
v0144com note when running this does not have any over head in terms
v0145com of slowing the monitor since no interrupts are involved
v0146com the clock is limited to about one minute of elapsed time
v0147com channel 0 will be used for the clock at factory installed
v0148com address of f0
v0149com channel 1 is used to store current time at a factory installed
v0150com address of f1
v0151incl h.clock (:)
v0152begin text
v0153jp @@<arnd> ;3m 10t 3b
v0154@i<initlk>;
v0155endtext
v0156calc initlk=initlk+1
v0157begin text
v0158ld a,0010100b;2m 7t 1b counter 0 + load lsb then mb+ mode2+ bcd
v0159out (0f3h),a ;3m 11t 2b set mode control

```



```

v01601d a,00 ;2m 7t 1b lsb of 2000 bcd
v0161out (0f0h),a ;3m 11t 2b load counter time channel 0
v01621d a,20h ;2m 7t 1b msb of 2000 bcd
v0163out (0f0h),a ;3m 11t 2b load counter time channel 0
v01641d a,0110001b;2m 7t 1b counter 1 + load lsb then mb+ mode2+ hex
v0165out (0f3h),a ;3m 11t 2b set mode control
v01661d a,00 ;2m 7t 1b lsb of ffff+1 hex
v0167out (0f1h),a ;3m 11t 2b load counter time channel 1
v01681d a,00h ;2m 7t 1b msb of ffff+1 hex
v0169out (0f1h),a ;3m 11t 2b load counter time channel 1
v0170jp @i<initlk> ;3m 10t 3b
v0171@<arnd>;nop ;1m 4t 3b isolate the initialization
v0172endtext
v0173calc arnd=arnd+1
v0174calc romptr=romptr+27
v0175s.and (rslt,arg1,arg2:0,8,0,8,0,8:11,47,14,10,0,175,185)
v0176com primitive to perform logical and
v0177com list=result,argument 1, argument 2 :stor,time,ext,c,i,addrs
v0178begin stext
v01791d a,<arg1> ;4m 13t 3b rslt = arg1 .and. arg2
v01801d b, a ;1m 4t 1b
v01811d a,<arg2> ;4m 13t 3b
v0182and b ;1m 4t 1b
v01831d (<rslt>),a ;4m 13t 3b
v0184endtext
v0185calc romptr=romptr+11
v0186s.add (rslt,arg1,arg2:0,16,0,16,0,16:13,71,21,9,0,186,195)
v0187com primitive to add arg1 and arg2 and store in rslt
v0188com list=rslt,arg1,arg2:precisions:s,t,e,c,i,addr
v0189begin stext
v01901d hl, (<arg1>);6m 20t 4b load arg1 in hl pair
v01911d bc, (<arg2>);6m 20t 4b load arg2 in bc pair
v0192add hl, bc ;3m 11t 1b add
v01931d (<rslt>),hl;6m 20t 4b save result
v0194endtext
v0195calc romptr=romptr+13
v0196s.conv816 (rslt,arg1:0,16,0,8:17,68,20,11,0,196,207)
v0197com routine to convert a 8 bit to 16 bit number
v0198begin stext
v01991d h, 0 ;2m 7t 2b clear high byte
v02001d a,<arg1> ;4m 13t 3b load 8 bit number
v02011d l, a ;1 4t 1b convert 8 to 16 bit
v0202add a, 80h ;2m 7t 2b check to see if negative
v0203jp nc, $+5 ;3m 10t 3b
v02041d h, 0ffh ;2m 7t 2b negative
v02051d (<rslt>),hl;6m 20t 4b save result
v0206endtext
v0207calc romptr=romptr+17
v0208s.float (rslt,arg1:,0,32,0,16:46,528,143,59,0,208,267)
v0209com primitive to convert an 16 bit number to floating point
v0210com normalization to 2's complement form exp mantissa
v0211com exponent is also in 2's complement form.
v0212com mantissa is normalized to sign.magnitude

```



```

v0213com list=rslt,arg1:precisions:s,t,e,c,i,addr
v0214begin stext
v0215ld hl,(<arg1>);6m 20t 4b 1b load arg1
v0216exx ;1m 4t 1b prime registers
v0217ld hl, 0 ;3m 10t 3b zero byte 3and4
v0218ld c, 15 ;2m 7t 2b put 14 in c' for largest exponent
v0219exx ;1m 4t 1b return to main registers
v0220ld a, 0 ;2m 8t 2b put zero in accumulator -- zero rslt?
v0221cp h ;1m 4t 1b is msb zero
v0222jr nz, $+011h ;2m 7t 2b quit test of rslt if non zero
v0223cp l ;1m 4t 1b is byte2 zero
v0224jr nz, $+00eh ;2m 7t 2b quit test of rslt if non zero
v0225exx ;am 4t 1b prime registers
v0226cp h ;1m 4t 1b is byte 2 zero
v0227jr nz, $+009h ;2m 7t 2b quit test of rslt and flip reg non zero
v0228cp l ;1m 4t 1b is byte 3 zero
v0229jr nz, $+006h ;2m 7t 2b quit test of rslt and flip reg non zero
v0230ld c, a ;1m 4t 1b put 0 in exponent result is zero
v0231exx ;1m 4t 1b main registers arg2 is 0
v0232jr $+031h ;3m 12t 2b result is zero store
v0233exx ;1m 4t 1b main registers
v0234ld a,h ;1m 4t 1b test sign of result
v0235and a ;1m 4t 1b set sign bit
v0236jp m,$+017h ;3m 10t 3b if neg goto neg normalization
v0237bit 6, h ;2m 8t 2b test ms bit s1.xxxx pos normal
v0238jp nz,$+027h ;3m 10t 3b if 1 in msb then store result
v0239exx ;1m 4t 1b prime registers
v0240cp c ;1m 4t 1b is exponent zero?
v0241jp z, $+021h ;3m 10t 3b if exponent is 0 then stop
v0242dec c ;1m 4t 1b decrement exponent in c'
v0243add hl, hl ;3m 11t 1b shift rslt low word
v0244exx ;1m 4t 1b main registers
v0245rl l ;2m 8t 2b shift rslt high word
v0246rl h ;2m 8t 2b
v0247jp $-011h ;3m 10t 3b check to see if normalized
v0248bit 6, h ;2m 8t 2b test ms bit s0.xxxx neg normal
v0249jp z, $+013h ;3m 10t 3b if 0 in msb then store result
v0250exx ;1m 4t 1b prime registers
v0251cp c ;1m 4t 1b is exponent zero?
v0252jp z, $+00dh ;3m 10t 3b if exponent is 0 then stop
v0253dec c ;1m 4t 1b decrement exponent in c'
v0254add hl, hl ;3m 11t 1b shift rslt low word
v0255exx ;1m 4t 1b main registers
v0256rl l ;2m 8t 2b shift rslt high word
v0257rl h ;2m 8t 2b
v0258jp $-011h ;3m 10t 3b check to see if normalized
v0259exx ;1m 4t 1b main registers
v0260ld (<rslt+1>),hl;5m 16t 3b save rslt ms word
v0261exx ;1m 4t 1b prime registers
v0262ld (<rslt+3>),hl;5m 16t 3b save rslt ls word
v0263ld a, c ;1m 4t 1b get exponent of rslt
v0264ld (<rslt>),a ;4m 13t 3b save rslt exponent
v0265exx ;1m 4t 1b main registers

```



```

v0266endtext
v0267calc romptr=romptr+34
v0268s.xor      (rslt,arg1,arg2:0,8,0,8,0,8:11,47,14,10,0,268,278)
v0269com primitive to perform logical exclusive or
v0270com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0271begin text
v02721d a,(<arg1>) ;4m 13t 3b      rslt = arg1 .xor. arg2
v02731d b, a ;1m 4t 1b
v02741d a,(<arg2>) ;4m 13t 3b
v0275xor b ;1m 4t 1b
v02761d (<rslt>),a ;3m 13t 3b
v0277endtext
v0278calc romptr=romptr+11
v0279s.rdtme      (rslt:0,16:10,58,16,16,0,279,295)
v0280com primitive to read a clock generated by h.clock.
v0281com current time is stored in the channel 1 counter of the cpu
v0282com ctc chip. to access this particular time rdtme must be used
v0283com the clock is limited to about one minute of elapsed time
v0284com channel 0 will be used for the clock at factory installed
v0285com address of f0
v0286com channel 1 is used to store current time at a factory installed
v0287com address of f1
v0288begin text
v02891d a,0100001b;2m 7t 1b counter 1 + latch + mode2+ hex
v0290out (0f3h),a ;3m 11t 2b set mode control to latch channel 1
v0291in l,(0f1h) ;3m 12t 2b get lsb
v0292in h,(0f1h) ;3m 12t 2b get msb
v02931d (<rslt>),h;5m 16t 3b load time to <rslt>
v0294endtext
v0295calc romptr=romptr+10
v0296s.outled      (arg1:0,8:24,100,25,10,8,296,331)
v0297com primitive to add output routine for up to 8 light emitting
v0298com diods on the 7303 keyboard/display board.
v0299com keybrd is a boolean flag indicating if board has previously been
v0300com called.
v0301com nled is the number of the outled that have been requested.
v0302com arg1 is the boolean that is tested to determine if the led should
v0303com be lighted.
v0304incl h.keydisplay(;)
v0305if nled .ne.-1 skip 6
v0306calc romptr=ramptr-1
v0307begin text
v0308org <ramptr>
v0309@outled: defb 0 ; set status of all lights off
v0310org <romptr>
v0311endtext
v0312calc nled=nled+1
v0313if nled .lt. 8 skip 4
v0314begin text
v0315you have requested more than 8 leds on the keyboard display
v0316the board is limited to a maximum of 8
v0317endtext
v0318begin text

```



```

v0319      ;<arg1> is displayed via lamp led number <nled>
v0320ld a, 0      2m 7t 2b      write inhibit the alphanumeric display
v0321out (0d1h),a      3m 11t 2b      send it to control port
v0322ld a,<arg1>      3m 13t 3b      see if light should be on
v0323and a      1m 4t 1b      set flags
v0324ld a,<outled>      3m 13t 3b      recall what status of all lights are
v0325res <nled>,a      2m 8t 2b
v0326jp z,$+5      3m 10t 3b
v0327set <nled>,a      2m 8t 2b
v0328ld <outled>,a;3m 13t 3b      save status of lamps
v0329out (0d0h),a      3m 13t 3b      light appropriate lamp
v0330endtext
v0331calc romptr=romptr+24
v0332s.not      (rslt,arg1:0,8,0,8,0,8,7,30,9,8,0,332,340)
v0333com primitive to perform logical not, complement
v0334com list=result,argument 1, argument 2 ::stor,time,ext,c,i,adds
v0335begin stext
v0336ld a,<arg1>      4m 13t 3b      rslt = not arg1
v0337cpl      1m 4t 1b
v0338ld <rslt>,a      4m 13t 3b
v0339endtext
v0340calc romptr=romptr+7
v0341s.eq      (rslt,arg1,arg2:0,8,0,8,0,8,15,72,20,13,0,341,354)
v0342com primitive to perform comparison between 2 8-bit numbers
v0343com list=result,argument 1, argument 2 ::stor,time,ext,c,i,adds
v0344begin stext
v0345ld a,<arg1>      4m 13t 3b      if arg1 = arg2 then rslt=ffh
v0346ld b,a      1m 4t 1b
v0347ld a,<arg2>      4m 13t 3b
v0348cp b      1m 4t 1b
v0349ld a,1111111b;2m 8t 1b
v0350jr z,$+3      3m 13t 2b      result equal
v0351cpl      1m 4t 1b      result not equal
v0352ld <rslt>,a      4m 13t 3b
v0353endtext
v0354calc romptr=romptr+15
v0355s.sub      (rslt,arg1,arg2:0,16,0,16,0,16,15,79,23,10,0,355,365)
v0356com primitive to subtract arg2 from arg1 and store answer in rslt
v0357com list=result,arg1,arg2:precisions:s,t,e,c,i,addr
v0358begin stext
v0359and a      1m 4t 1b      clear carry flag
v0360ld hl,<arg1>      6m 20t 4b      load arg1 in hl pair
v0361ld bc,<arg2>      6m 20t 4b      load arg2 in bc pair
v0362sbc hl,bc      4m 15t 2b      subtract with carry
v0363ld <rslt>,hl      6m 20t 4b      save result
v0364endtext
v0365calc romptr=romptr+15
v0366s.lt      (rslt,arg1,arg2:0,8,0,8,0,8,45,118,34,29,0,366,395)
v0367com primitive to perform comparison between 2 8-bit numbers
v0368com list=result,argument 1, argument 2 ::stor,time,ext,c,i,adds
v0369begin stext
v0370ld a,<arg1>      4m 13t 3b      if arg1 lt arg2 then rslt=ffh
v0371ld b,a      1m 4t 1b      b=arg1

```


v0372ld	a,(<arg2>)	:4m	13t	3b	
v0373ld	c, a	:1m	4t	1b	c=arg2
v0374and	a	:1m	4t	1b	set sign flag of arg2
v0375jp	p,\$+00dh	:3m	10t	3b	jump if arg2 is positive
v0376ld	a, b	:1m	4t	1b	arg2 = -
v0377and	a	:1m	4t	1b	set sign flag of arg1
v0378ld	b, c	:1m	4t	1b	arg1 .swap. arg2
v0379jp	m,\$+011h	:3m	10t	3b	arg1 = - arg2 = - comp backwards
v0380ld	a, 0	:2m	7t	2b	arg1 = + arg2 = - false
v0381jr	\$+016h	:3m	12t	2b	
v0382ld	a, b	:1m	4t	1b	
v0383and	a	:1m	4t	1b	set sign flag of arg1
v0384ld	a, c	:1m	4t	1b	restore arg2 to accumulator
v0385jp	p,\$+007h	:3m	10t	3b	arg1 = + arg2 = +
v0386ld	a,11111111b	:2m	7t	2b	arg1 = - arg2 = + true
v0387jr	\$+00ch	:3m	12t	2b	
v0388cp	b	:1m	4t	1b	result false arg1 >= arg2
v0389ld	a,00000000b	:2m	7t	2b	
v0390jp	z, \$+7	:3m	10t	3b	
v0391jp	m, \$+4	:3m	10t	3b	result true arg1 lt arg2
v0392cpl		:1m	4t	1b	
v0393 ld	(<rs1t>),a	:4m	13t	3b	
v0394endtext					
v0395calc	romptr=romptr+19				
v0396s.ne	(rs1t,arg1,arg2:0,8,0,8,0,8:15,72,20,13,0,396,409)				
v0397com	primitive to perform comparison between 2 8-bit numbers				
v0398com	list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs				
v0399begin stext					
v0400ld	a,(<arg1>)	:4m	13t	3b	if arg1 arg2 then rs1t=ffh
v0401ld	b, a	:1m	4t	1b	
v0402ld	a,(<arg2>)	:4m	13t	3b	
v0403cp	b	:1m	4t	1b	
v0404ld	a, 0	:2m	8t	1b	
v0405jr	z, \$+003h	:3m	13t	2b	result not equal
v0406cpl		:1m	4t	1b	result equal
v0407ld	(<rs1t>),a	:4m	13t	3b	
v0408endtext					
v0409calc	romptr=romptr+15				
v0410s.ge	(rs1t,arg1,arg2:0,8,0,8,0,8:42,108,31,28,0,410,438)				
v0411com	primitive to perform comparison between 2 8-bit numbers				
v0412com	list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs				
v0413begin stext					
v0414ld	a,(<arg2>)	:4m	13t	3b	if arg2 le arg1 then rs1t=ffh
v0415ld	b, a	:1m	4t	1b	b=arg2
v0416ld	a,(<arg1>)	:4m	13t	3b	
v0417ld	c, a	:1m	4t	1b	c=arg1
v0418and	a	:1m	4t	1b	set sign flag of arg1
v0419jp	p,\$+00dh	:3m	10t	3b	jump if arg1 is positive
v0420ld	a, b	:1m	4t	1b	arg1 = -
v0421and	a	:1m	4t	1b	set sign flag of arg2
v0422ld	b, c	:1m	4t	1b	arg2 .swap. arg1
v0423jp	m, \$+011h	:3m	10t	3b	arg2 = - arg1 = - comp backwards
v0424ld	a, 0	:2m	7t	2b	arg2 = + arg1 = - false


```

v0425jrc $+013h ;3m 12t 2b
v0426ld a, b ;1m 4t 1b
v0427and a ;1m 4t 1b
v0428ld a, c ;1m 4t 1b
v0429jp p, $+007h ;3m 10t 3b
v0430ld a,11111111b;2m 7t 2b
v0431jrc $+009h ;3m 12t 2b
v0432cp b ;1m 4t 1b
v0433ld a,11111111b;2m 7t 2b
v0434jp p, $+4 ;3m 10t 3b
v0435cpl ;1m 4t 1b
v0436ld (<rslt>),a ;4m 13t 3b
v0437endtext
v0438calc romptr=romptr+42
v0439s.rocker (arg1:0,8:13,53,14,7,6,439,461)
v0440com primitive to add a test of the rocker switches on the 7303
v0441com keyboard/display board. it is limited to two switches.
v0442com an error is generated if more than two switches are requested.
v0443com nrocker is the number of rocker switches that have been requested
v0444com arg1 is a boolean value based on the value of the rocker switch.
v0445incl h.keydisplay(;;)
v0446calc nrocker=nrocker + 1
v0447if nrocker .lt. 3 skip 4
v0448begin text
v0449you have requested more than 2 rocker switcher on the keyboard/display.
v0450the board is limited to a maximum of 2
v0451endtext
v0452calc scrtch= 5 + nrocker
v0453begin text
v0454 in a,(0d0h) ;3m 11t 2b get status of switch <nrocker>
v0455 bit <scrtch>,a ;2m 8t 2b set carry flag if true
v0456 ld a, 0 ;2m 7t 2b assume switch is off
v0457 jp nc, $+4 ;3m 10t 3b if <nrocker> is off set <arg1> false
v0458 cpl ;1m 4t 1b change result to true
v0459 ld (<arg1>),a ;3m 13t 3b return the status of switch to <arg1>
v0460endtext
v0461calc romptr=romptr+13
v0462s.forend (indx,slab,elab:0,8,0,255:7,27,8,9,0,462,471)
v0463com primitive to end a for loop
v0464com list=indx,start label,end label
v0465com max loop count 255
v0466begin text
v0467ld a,<indx> ;4m 13t 3b get value of index at top of loop
v0468inc a ;1m 4t 1b crank index
v0469elab:jp <slab> ;3m 10t 3b jump to for loop test
v0470endtext
v0471calc romptr=romptr+7
v0472s.le (rslt,arg1,arg2:0,8,0,8,0,8,0,42,108,31,28,0,472,500)
v0473com primitive to perform comparison between 2 8-bit numbers
v0474com list=result,argument 1, argument 2 ::stor,time,ext,c,i,adds
v0475begin text
v0476ld a,<arg1> ;4m 13t 3b if arg1 le arg2 then rslt=ffh
v0477ld b, a ;1m 4t 1b b=arg1

```



```

v04781d a,(<arg2>) ;4m 13t 3b
v04791d c, a ;1m 4t 1b
v0480and a ;1m 4t 1b
v0481jp p,$+00dh ;3m 10t 3b
v04821d a, b ;1m 4t 1b
v0483and a ;1m 4t 1b
v04841d b, c ;1m 4t 1b
v0485jp m, $+011h ;3m 10t 3b
v04861d a, 0 ;2m 7t 2b
v0487jr $+013h ;3m 12t 2b
v04881d a, b ;1m 4t 1b
v0489and a ;1m 4t 1b
v04901d a, c ;1m 4t 1b
v0491jp p, $+007h ;3m 10t 3b
v04921d a,1111111b;2m 7t 2b
v0493jr $+009h ;3m 12t 2b
v0494cp b ;1m 4t 1b
v04951d a,1111111b;2m 7t 2b
v0496jp p, $+4 ;3m 10t 3b
v0497cpl ;1m 4t 1b
v04981d (<rslt>),a ;4m 13t 3b
v0499endtext
v0500calc romptr=romptr+42
v0501s.addck (rslt,arg1,arg2:0,8,0,8,0,8,23,80,24,14,0,501,515)
v0502com primitive to add arg1 and arg2 and store in rslt
v0503com list=rslt,arg1,arg2:precisions:s,t,e,c,i,addr
v0504begin stext
v05051d a, (<arg1>) ;13t 4m 3b store arg1 in accumulator
v05061d hl,<arg2> ;10t 3m 3b have hl point to arg2 byte
v0507add a, (hl) ;7t 2m 1b add accumulator with arg2
v0508jp po, $+13 ;3m 10t 3b if no overflow store result
v0509jp c, $+8 ;3m 10t 3b if carry the maximize minus rslt
v05101d a,0111111b;2m 7t 2b put in largest positive value
v0511jp $+5 ;3m 10t 3b
v05121d a,10000000b ;2m 7t 2b put in largest negative value
v05131d (<rslt>),a ;13t 4m 3b save result of add in rslt
v0514endtext
v0515calc romptr=romptr+23
v0516s.sub (rslt,arg1,arg2:0,8,0,8,0,8,10,43,13,9,0,516,525)
v0517com primitive to subtract arg2 from arg1 and store in rslt
v0518com list= rslt,arg1,arg2:precisions:s,t,e,c,i,a
v0519begin stext
v05201d a,(<arg1>) ;4m 13t 3b load arg1 in accumulator
v05211d hl,<arg2> ;3m 10t 3b point hl to arg2
v0522sub (hl) ;2m 7t 1b arg1 - arg2
v05231d (<rslt>), a ;4m 13t 3b save result
v0524endtext
v0525calc romptr=romptr+10
v0526s.add (rslt,arg1,arg2:0,8,0,8,0,8,10,43,13,9,0,526,535)
v0527com primitive to add arg1 and arg2 and store in rslt
v0528com list=rslt,arg1,arg2:precisions:s,t,e,c,i,addr
v0529begin stext
v05301d a, (<arg1>) ;13t 4m 3b store arg1 in accumulator

```



```

v0531ld hl,<arg2> ;10t 3m 3b have hl point to arg2 byte
v0532add a,(hl) ;7t 2m 1b add accumulator with arg2
v0533ld (<rslt>),a ;13t 4m 3b save result of add in rslt
v0534endtext
v0535calc romptr=romptr+10
v0536s.lt (rslt,arg1,arg2:0,8,0,16,0,16:46,131,38,26,0,536,562)
v0537com primitive to perform comparison between 2 16-bit numbers
v0538com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0539begin stext
v0540ld de,<arg1> ;6m 20t 4b if arg1 lt arg2 then rslt=ffh de=<arg1>
v0541ld hl,<arg2> ;5m 16t 3b hl=<arg2>
v0542ld a,h ;1m 4t 1b
v0543and a,h ;1m 4t 1b
v0544jp p,$+13 ;3m 10t 3b set sign flag of arg2
v0545ld a,d ;1m 4t 1b jump if arg2 is positive
v0546and a ;1m 4t 1b arg2 = -
v0547jp m,$+18 ;3m 10t 3b set sign flag of arg1
v0548ld a,0 ;2m 7t 2b arg1 = - arg2 = - comp backwards
v0549jp $+24 ;3m 10t 3b arg1 = + arg2 = - false
v0550ld a,d ;1m 4t 1b
v0551and a ;1m 4t 1b
v0552jp p,$+8 ;3m 10t 3b set sign flag of arg1
v0553ld a,11111111b;2m 7t 2b arg1 = + arg2 = +
v0554jp $+14 ;3m 10t 3b arg1 = - arg2 = + true
v0555sbc hl,de ;4m 15t 2b
v0556ld a,00000000b;2m 7t 2b result false arg1 >= arg2
v0557jp z,$+7 ;3m 10t 3b
v0558jp m,$+4 ;3m 10t 3b result true arg1 lt arg2
v0559cpl ;1m 4t 1b
v0560ld (<rslt>),a ;4m 13t 3b
v0561endtext
v0562calc romptr=romptr+46
v0563s.forstart (indx,lwr,upr,slab,elab:0,8,0,8,0,8,255:13,50,15,11,0,563,574)
v0564com primitive to set up a loop with constant bounds
v0565com list=index,lower bound,upper bound,start label,end label
v0566com max loop count 255
v0567begin stext
v0568ld a,<lwr> ;2m 7t 2b lower bound of counter
v0569ld (<indx>),a ;4m 13t 3b initialize index to lower
v0570slab:ld a,<indx>;4m 13t 3b get value of index at top of loop
v0571cpl upr ;2m 7t 2b compare to upper limit
v0572jp z,<elab>+3 ;3m 10t 3b jump out of loop on index=upr
v0573endtext
v0574calc romptr=romptr+13
v0575s.ge (rslt,arg1,arg2:0,8,0,16,0,16:45,118,34,26,0,575,601)
v0576com primitive to perform comparison between 2 16-bit numbers
v0577com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0578begin stext
v0579ld de,<arg2> ;6m 20t 4b if arg2 learg1 then rslt=ffh de=<arg2>
v0580ld hl,<arg1> ;5m 16t 3b hl=<arg1>
v0581ld a,h ;1m 4t 1b
v0582and a,h ;1m 4t 1b set sign flag of arg1
v0583jp p,$+13 ;3m 10t 3b jump if arg1 is positive

```



```

v0584ld a, d ;1m 4t
v0585and a ;1m 4t
v0586jp m,$+18 3b
v0587ld a, 0 ;3m 10t
v0588jp $+24 ;2m 7t
v0589ld a, d ;3m 10t 3b
v0590and a ;1m 4t 1b
v0591jp p,$+8 ;1m 4t
v0592ld a,11111111b;2m 7t
v0593jp $+14 ;3m 10t 3b
v0594sbc hl,de ;4m 15t 2b
v0595ld a,00000000b;2m 7t
v0596jp m, $+7 ;3m 10t
v0597jp m, $+4 ;3m 10t
v0598cpl ;1m 4t
v0599 ld (<rslt>),a ;4m 13t
v0600endtext
v0601calc romptr=romptr+19
v0602s.eq (rslt,arg1,arg2:0,8,0,16,0,16:16,89,25,12,0,602,614)
v0603com primitive to perform comparision between 2 16-bit numbers
v0604com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0605begin text
v0606ld de,(<arg1>) ;6m 20t 4b if arg1 = arg2 then rslt=ffh de=<arg1>
v0607ld hl,(<arg2>) ;5m 16t 3b hl=<arg2>
v0608sbc hl,de ;4m 15t 2b
v0609ld a,11111111b;2m 8t 1b
v0610jr z, $+3 ;3m 13t 2b result equal
v0611cpl ;1m 4t 1b result not equal
v0612 ld (<rslt>),a ;4m 13t 3b
v0613endtext
v0614calc romptr=romptr+16
v0615s.le (rslt,arg1,arg2:0,8,0,16,0,16:45,118,34,26,0,615,641)
v0616com primitive to perform comparision between 2 16-bit numbers
v0617com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0618begin text
v0619ld de,(<arg1>) ;6m 20t 4b if arg1 learg2 then rslt=ffh de=<arg1>
v0620ld hl,(<arg2>) ;5m 16t 3b hl=<arg2>
v0621ld a, h ;1m 4t 1b
v0622and a ;1m 4t
v0623jp p,$+13 ;3m 10t
v0624ld a, d ;1m 4t
v0625and a ;1m 4t
v0626jp m,$+18 ;3m 10t
v0627ld a, 0 ;2m 7t
v0628jp $+24 ;3m 10t 3b
v0629ld a, d ;1m 4t 1b
v0630and a ;1m 4t
v0631jp p,$+8 ;3m 10t
v0632ld a,11111111b;2m 7t
v0633jp $+14 ;3m 10t 3b
v0634sbc hl,de ;4m 15t 2b
v0635ld a,00000000b;2m 7t
v0636jp m, $+7 ;3m 10t

```



```

v0637jp m, $+4 ;3m 10t 3b
v0638cpl ;1m 4t 1b
v0639 ld (<rs!t>),a ;4m 13t 3b
v0640endtext
v0641calc romptr=romptr+19
v0642s.gt (rs!t,arg1,arg2:0,8,0,16,0,16:45,118,34,26,0,642,668)
v0643com primitive to perform comparision between 2 16-bit numbers
v0644com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0645begin stext
v0646ld de,(<arg2>) ;6m 20t 4b if arg2 lt arg1 then rs!t=ffh de=<arg2>
v0647ld hl,(<arg1>) ;5m 16t 3b hl=<arg1>
v0648ld a, h ;1m 4t 1b
v0649and a ;1m 4t 1b
v0650jp p,$+13 set sign flag of arg1
v0651ld a, d ;3m 10t 3b jump if arg1 is positive
v0652and a ;1m 4t 1b arg1 = -
v0653jp m,$+18 set sign flag of arg2
v0654ld a, 0 ;3m 10t 3b arg2 = - arg1 = - comp backwards
v0655jp $+24 arg2 = + arg1 = - false
v0656ld a, d ;3m 10t 3b
v0657and a ;1m 4t 1b
v0658jp p,$+8 set sign flag of arg2
v0659ld a,11111111b;2m 7t 3b arg2 = + arg1 = +
v0660jp $+14 arg2 = - arg1 = + true
v0661sbc hl,de ;3m 10t 3b
v0662ld a,00000000b;2m 7t 2b result false arg2 gt arg1
v0663jp z, $+7 ;3m 10t 3b
v0664jp m, $+4 ;3m 10t 3b
v0665cpl ;1m 4t 1b
v0666 ld (<rs!t>),a ;4m 13t 3b result true arg2 le arg1
v0667endtext
v0668calc romptr=romptr+19
v0669s.clockcon25(::27,14,4,17,12,669,704)
v0670com primitive to generate a 25 millisecond clock. the value of
v0671com current time is stored in the channel 1 counter of the cpu
v0672com ctc chip. to access this particular time rdtime must be used
v0673com to read/rest the time use rdrstime primitive.
v0674com note when running this does not have any over head in terms
v0675com of slowing the monitor since no interrupts are involved
v0676com the clock is limited to about one minute of elapsed time
v0677com channel 0 will be used for the clock at factory installed
v0678com address of f0
v0679com channel 1 is used to store current time at a factory installed
v0680com address of f1
v0681incl h.clock (:)
v0682begin stext
v0683jp @w<arnd> ;3m 10t 3b
v0684@i<initlk>:
v0685endtext
v0686calc initlk=initlk+1
v0687begin stext
v0688ld a,00110101b;2m 7t 1b counter 0 + load lsb then mb+ mode2+ hex
v0689out (0f3h),a ;3m 11t 2b set mode control

```



```

v06901d a,050h      ;2m 7t 1b lsb of 50000 decimal or c350 hex
v0691out (0f0h),a    ;3m 11t 2b load counter time channel 0
v06921d a,0c3h      ;2m 7t 1b msb of 2000 bcd
v0693out (0f0h),a    ;3m 11t 2b load counter time channel 0
v06941d a,0110001b;2m 7t 1b counter 1 + load lsb then mb+ mode2+ hex
v0695out (0f3h),a    ;3m 11t 2b set mode control
v06961d a,00        ;2m 7t 1b lsb of ffff+1 hex
v0697out (0f1h),a    ;3m 11t 2b load counter time channel 1
v06981d a,00h       ;2m 7t 1b msb of ffff+1 hex
v0699out (0f1h),a    ;3m 11t 2b load counter time channel 1
v0700jp @i<initlk>  ;3m 10t 3b
v0701@<arnd>;nop ;1m 4t 3b isolate the initialization
v0702endtext
v0703calc arnd=arnd+1
v0704calc romptr=romptr+27
v0705s.main ( ;:12,44,12,5,3,705,764)
v0706com primitive to define controller setup and initialization
v0707com list = empty : empty : storage, time, ext, calc, incl, addr
v0708incl h.processor (:)
v0709incl h.cardcage (:)
v0710calc ramptr=36767
v0711calc romptr=102
v0712calc romptr=16384
v0713calc ramptr=ramptr-1
v0714calc ramptr=ramptr-32
v0715com the rom pointer is set to start at 102 or 66h corresponds to reset
v0716com location for the z-80. this will be the cold boot point for the
v0717com controller. all initializations will be done immediately after
v0718com this point. following the initializations will be the top of the
v0719com polling loop for the task contingency pairs.
v0720com to allow the use of a debug prom developed at the naval
v0721com postgraduate school electrical engineering department the
v0722com starting location is changed to 4096 to allow a the system
v0723com to auto boot and to allow loading of memory from another
v0724com computer via the dual uart card. the loading prom inhibits the
v0725com use of the reset location because of the location of the code and
v0726com the interrupt loactions used in a debugger for the prolog system
v0727begin stext
v0728;
v0729; zilog z-80 based system
v0730;
v0731; idsec
v0732; idsec
v0733; idsec
v0734;
v0735;
v0736.z80
v0737aseg
v0738endtext
v0739if debug .eq. 0 skip 4
v0740begin stext
v0741org 100h
v0742start:jp 4000h ;start execution at 100h then jump to 4000h offset

```



```

v0743endtext
v0744begin text
v0745org <ramptr>
v0746@initvar: defb 0
v0747@stak: defb 32
v0748org <romptr>
v0749@cold: ld sp, @stak+32
v0750ld a, 0
v0751ld (@initvar), a
v0752di
v0753endtext
v0754if debug .eq. 1 skip 3
v0755begin text
v0756jp @i<initlk>
v0757endtext
v0758if debug .eq. 0 skip 5
v0759begin text
v0760jp @spvsr
v0761
v0762endtext
v0763incl s.wrtbin
v0764calc romptr=romptr+12
v0765s.inkey (arg1:0,8:11,47,14,11,0,765,776)
v0766com primitive to add input routine for keyboard entry using the
v0767com 7303 keyboard/display board.
v0768com keybrd is a boolean flag indicating if board has previously been
v0769com called.
v0770com nkey is the number of the inkeys that have been requested.
v0771com arg1 is the boolean that is toggled by the keyboard. pressing
v0772com the appropriate key will cause the value of the boolean to be
v0773com complemented.
v0774begin text
v0775endtext
v0776calc romptr=romptr+11
v0777s.outdigit (arg1:0,8:11,47,14,0,0,777,789)
v0778com primitive to add output routines to display a message to the
v0779com alphanumeric display on the 7303 keyboard/display board.
v0780com keybrd is a boolean flag indicating if board has previously been
v0781com called.
v0782com nodgt is a boolean flag indicating that the outdigit routines have
v0783com been previously added. they will be called as a subroutine to
v0784com display the output. there is no limit to the number of different
v0785com messages that can be displayed.
v0786com arg1 is the boolean that is tested to determine if the led should
v0787com be lighted.
v0788begin text
v0789endtext
v0790s.consfp (nam,val, :0,32:5,0,0,8,0,790,798)
v0791com primitive to define data for floating point number
v0792com list=data-name,value=value-prec,stor,time,ext,c,i,addr
v0793begin text
v0794<nam>: defb <val> ;define a one byte integer exponent
v0795 defw <man1> ;define msw of mantissa

```



```

v0796      defw  <man2> ;define lsw of mantissa
v0797endtext
v0798calc romptr=romptr+5
v0799s.addck  (rslt,arg1,arg2:0,16,0,16,0,16:33,144,39,18,0,799,817)
v0800com primitive to add arg1 and arg2 and store in rslt
v0801begin stext
v0802ld hl, (<arg1>);6m      20t      4b      load arg1 in hl pair
v0803ld bc, (<arg2>);6m      20t      4b      load arg2 in bc pair
v0804ld a, l ;1m             4t      1b
v0805add a, c ;1m            4t      1b      add lsb
v0806ld l, a ;1m             4t      1b
v0807ld a, h ;1m             4t      1b
v0808adc a, b ;1m            4t      1b      add msb
v0809ld h, a ;1m             4t      1b
v0810jp po, $+15 ;3m          10t      3b      if no overflow store result
v0811jp c, $+9 ;3m            10t      3b      if carry the maximize minus rslt
v0812ld hl, 7fffh ;3m         10t      3b      put in largest positive value
v0813jp $+6 ;3m              10t      3b
v0814ld hl, 8000h ;3m         10t      3b      put in largest negative value
v0815ld (<rslt>),hl;6m        20t      4b      save result
v0816endtext
v0817calc romptr=romptr+33
v0818s.assigncons(var,data:0,8,0,8:5,20,6,7,0,818,825)
v0819com primitive to assign a value of constant to a variable
v0820com list=var,data-var:var-prec,data-prec:stor,time,ext,calc,incl,addr
v0821begin stext
v0822ld a,<data> ;2m          7t      2b      assign <data>
v0823ld (<var>),a ;4m         13t      3b      to <var>
v0824endtext
v0825calc romptr=romptr + 5
v0826s.assign (var,data:0,8,0,8:6,26,8,7,0,826,833)
v0827com primitive to assign a value of one variable to another variable
v0828com list=var,data-var:var-prec,data-prec:stor,time,ext,calc,incl,addr
v0829begin stext
v0830ld a,<data> ;4m          13t      3b      assign <data>
v0831ld (<var>),a ;4m         13t      3b      to <var>
v0832endtext
v0833calc romptr=romptr + 6
v0834s.assign (var,data:0,16,0,16:7,36,11,7,0,834,841)
v0835com primitive to assign a value of one variable to another variable
v0836com list=var,data-var:var-prec,data-prec:stor,time,ext,calc,incl,addr
v0837begin stext
v0838ld hl,<data> ;6m         20t      4b      assign <data>
v0839ld (<var>),hl ;5m        16t      3b      to <var>
v0840endtext
v0841calc romptr=romptr + 7
v0842s.proc (nam::1,4,1,7,0,842,849)
v0843com primitive to define procedure entry point
v0844com list=proc-name :empty:storage,time,ext,calc,incl,addr
v0845begin stext
v0846;procedure <nam>
v0847@<nam>: nop ;1m         4t      1b      entry point for <nam>
v0848endtext

```



```

v0849calc romptr=romptr+1
v0850s.cons      (nam,val, :0,8:1,0,0,6,0,850,856)
v0851com primitive to define data
v0852com list=data-name,value=value-prec,stor,time,ext,c,i,addr
v0853begin stext
v0854<nam>:      defb <val> ;reserve one byte for data
v0855endstext
v0856calc romptr=romptr+1
v0857s.tabent    (fnc,task :10,51,15,10,0,857,867)
v0858com primitive to add one entry to monitor table
v0859com list= func-name, task name:empty:s,t,e,c,i,address
v0860begin stext
v0861call @<fnc> ;5m 17t 3b test for contingency <fnc>
v0862ld a,<fnc> ;4m 13t 3b get contingency result
v0863cp 1111111b ;1m 4t 1b check if result true
v0864call z,<task> ;5m 17t 3b if true execute task
v0865          ;if not true get next tabent or tabend to loop
v0866endstext
v0867calc romptr=romptr+10
v0868s.tabaccp2 (:...0,0,868,879)
v0869com this is a dummy primitive to allow compatibility with the 8080
v0870com library. the functions that would be performed in this primitive
v0871com are all located in s.tabent. this has the effect of eliminating
v0872com intermediate table and increasing execution speed. if there are
v0873com wide variations in contingency/task speeds more memory will be
v0874com than in the 8080 primitive. note s.main is also changed because
v0875com of the elimination of the intermediate table
v0876com list= func-name, task name:empty:s,t,e,c,i,address
v0877begin stext
v0878          ; this space is deliberately void. this is a dummy primitive.
v0879endstext
v0880s.var      (name:0,8:0,0,0,3,0,880,888)
v0881com primitive to define storage for 8 bit variable integer or logical
v0882com list=data-name,value=value-prec,stor,time,ext,c,i,addr
v0883calc ramptr=ramptr - 1
v0884begin stext
v0885org <ramptr> ;8 bit variable <name> in ram
v0886<name>:      defb 0 ;0m 0t 1b
v0887org <romptr>
v0888endstext
v0889s.perform (name : 3,17,5,6,0,889,895)
v0890com primitive to invoke a procedure
v0891com list=proc-name :empty: storage,time,ext,calc,incl,addr
v0892begin stext
v0893call @<name> ;5m 17t 3b perform procedure <name>
v0894endstext
v0895calc romptr=romptr+3
v0896s.subck     (rslt,arg1,arg2:0,16,0,16,0,16:33,134,39,19,0,896,915)
v0897com primitive to subtract arg2 from arg1 and store answer in rslt
v0898com list=rslt,arg1,arg2:precisions:s,t,e,c,i,addr
v0899begin stext
v0900ld hl, (<arg1>) ;6m 20t 4b load arg1 in hl pair
v0901ld bc, (<arg2>) ;6m 20t 4b load arg2 in bc pair

```


v0902ld a, l	:1m	4t	1b	
v0903sub c	:1m	4t	1b	subtract lsb
v0904ld l, a	:1m	4t	1b	
v0905ld a, h	:1m	4t	1b	
v0906sbc a, b	:1m	4t	1b	subtract msb
v0907ld h, a	:1m	4t	1b	
v0908jp po, \$+15	:3m	10t	3b	if no overflow store result
v0909jp c, \$+9	:3m	10t	3b	if carry the maximize minus rslt
v0910ld hl, 7fffh	:3m	10t	3b	put in largest positive value
v0911jp \$+6	:3m	10t	3b	
v0912ld hl, 8000h	:3m	10t	3b	put in largest negative value
v0913ld (<rslt>),hl	:6m	20t	4b	save result
v0914endtext				
v0915calc romptr=romptr+33				
v0916s.fadd (rslt,arg1,arg2:0,32,0,32,0,32:205,2263,597,126,0,916,1042)				
v0917com primitive to add 2 floating point numbers and store rslt				
v0918com format exp,byte1,byte2,byte3,byte4				
v0919com mantissa is in form s1.xxxxx				
v0920com s is sign of number xxx is two's complement number				
v0921com time calculation is based on 31 shifts on entry				
v0922com list=rslt,arg1:precisions:s,t,e,c,i,addr				
v0923begin stext				
v0924ld de,(<arg1>+1):6m	20t	4b		first word
v0925ld hl,(<arg2>+1):5m	16t	3b		first word
v0926exx	:1m	4t	1b	prime registers
v0927ld de,(<arg1>+3):6m	20t	4b		second word
v0928ld hl,(<arg2>+3):5m	16t	3b		second word
v0929ld a,(<arg1>):4m	13t	3b		get exponent of arg1
v0930ld b, a	:1m	4t	1b	save exponent in b'
v0931ld a,(<arg2>):4m	13t	3b		get exponent of arg2
v0932ld c, a	:1m	4t	1b	save exponent in c'
v0933exx	:1m	4t	1b	main registers
v0934ld a, 10000000b;2m	8t	2b		mask for sign
v0935and d	:1m	4t	1b	sign of arg1
v0936ld b, a	:1m	4t	1b	save sign of arg1 main
v0937ld a, 10000000b;2m	8t	2b		mask for sign
v0938and h	:1m	4t	1b	sign of arg2
v0939ld c, a	:1m	4t	1b	save sign of arg2 main
v0940exx	:1m	4t	1b	prime registers
v0941ld a, b	:1m	4t	1b	put arg1 exponent in a
v0942sub c	:1m	4t	1b	subtract exponent arg2
v0943exx	:1m	4t	1b	main registers
v0944jp z,\$+04lh	:3m	10t	3b	exponents are equal no shift required
v0945jp m,\$+024h	:3m	10t	3b	arg1 lt arg2 exponent
v0946cp 3l	:2m	7t	2b	max number of shifts
v0947jp m,\$+005h	:3m	10t	3b	is shifts lt 31
v0948ld a, 3l	:2m	7t	2b	limit to 31 shifts
v0949sra h	:2m	8t	2b	shift msb arg2 right retain sign
v0950rr l	:2m	8t	2b	rotate byte2 arg2
v0951exx	:1m	4t	1b	prime registers
v0952rr h	:2m	8t	2b	rotate byte3 arg2
v0953rr l	:2m	8t	2b	rotate byte4 arg2
v0954jp nc,\$+005h	:3m	10t	3b	sticky bit

v0955set 0, 1	:2m	8t	2b	1b	set least significant bit
v0956exx	:1m	4t	1b	1b	return to main registers
v0957dec a	:1m	4t	1b	1b	
v0958jp nz, \$-010h	:3m	10t	3b	3b	check to see if exp are aligned
v0959ld c, b	:1m	4t	1b	1b	sign of result will be same as arg1
v0960exx	:1m	4t	1b	1b	prime registers
v0961ld c, b	:1m	4t	1b	1b	exponent for result is in c'
v0962exx	:1m	4t	1b	1b	main registers
v0963jp \$+01dh	:3m	10t	3b	3b	justified continue with add
v0964cp -31	:2m	7t	2b	2b	is shifts lt -31
v0965jp p, \$+005h	:3m	10t	3b	3b	is shifts lt 31
v0966ld a, -31	:2m	7t	2b	2b	limit to 31 shifts
v0967sra d	:2m	8t	2b	2b	shift msb arg1 right keep sign
v0968rr e	:2m	8t	2b	2b	rotate byte2 arg1
v0969exx	:1m	4t	1b	1b	prime registers
v0970rr d	:2m	8t	2b	2b	rotate byte3 arg1
v0971rr e	:2m	8t	2b	2b	rotate byte4 arg1
v0972jp nc, \$+005h	:3m	10t	3b	3b	sticky bit
v0973set 0, e	:2m	8t	2b	2b	set least significant bit
v0974exx	:1m	4t	1b	1b	main registers
v0975inc a	:1m	4t	1b	1b	add one to negative number till 0
v0976jp nz, \$-010h	:3m	10t	3b	3b	
v0977exx	:1m	4t	1b	1b	prime register beginning of add
v0978add hl, de	:3m	11t	1b	1b	add 3 and 4 bytes of mantissa
v0979exx	:1m	4t	1b	1b	main registers
v0980ld a, e	:1m	4t	1b	1b	prepare to add byte2
v0981adc a, l	:1m	4t	1b	1b	add byte2
v0982ld l, a	:1m	4t	1b	1b	save byte2
v0983ld a, d	:1m	4t	1b	1b	prepare to add byte1
v0984adc a, h	:1m	4t	1b	1b	add byte1
v0985ld h, a	:1m	4t	1b	1b	save byte1 of result
v0986jp pe, \$+006h	:3m	12t	1b	1b	correct for overflow
v0987jp \$+00eh	:3m	10t	3b	3b	no overflow
v0988rr h	:2m	8t	2b	2b	shift msb of mantissa of rslt right
v0989rr l	:2m	8t	2b	2b	rotate byte2
v0990exx	:1m	4t	1b	1b	prime registers
v0991rr h	:2m	8t	1b	1b	rotate byte3
v0992rr l	:2m	8t	2b	2b	rotate byte4
v0993inc c	:1m	4t	1b	1b	correct exponent in c'
v0994exx	:1m	4t	1b	1b	return to main registers
v0995ld a, 0	:2m	8t	2b	2b	put zero in accumulator -- zero rslt?
v0996cp h	:1m	4t	1b	1b	is msb zero
v0997jr nz, \$+011h	:2m	7t	2b	2b	quit test of rslt if non zero
v0998cp l	:1m	4t	1b	1b	is byte2 zero
v0999jr nz, \$+00eh	:2m	7t	2b	2b	quit test of rslt if non zero
v1000exx	:am	4t	1b	1b	prime registers
v1001cp h	:1m	4t	1b	1b	is byte 2 zero
v1002jr nz, \$+009h	:2m	7t	2b	2b	quit test of rslt and flip reg non zero
v1003cp l	:1m	4t	1b	1b	is byte 3 zero
v1004jr nz, \$+006h	:2m	7t	2b	2b	quit test of rslt and flip reg non zero
v1005ld c, a	:1m	4t	1b	1b	make exponent 0 in c'
v1006exx	:1m	4t	1b	1b	main registers arg2 is 0
v1007jr \$+031h	:3m	12t	2b	2b	result is zero store


```

v1008exx      ;1m 4t      1b      main registers
v1009ld a,h    ;1m 4t      1b      test sign of result
v1010and a      ;1m 4t      1b      set sign bit
v1011jp m,$+017h ;3m 10t    3b      if neg goto neg normalization
v1012bit 6,h    ;2m 8t      3b      test ms bit s1.xxxx pos normal
v1013jp nz,$+027h ;3m 10t    3b      if 1 in msb then store result
v1014exx      ;1m 4t      1b      prime registers
v1015cp c      ;1m 4t      1b      is exponent zero?
v1016jp z, $+021h ;3m 10t    3b      if exponent is 0 then stop
v1017dec c      ;1m 4t      1b      decrement exponent in c'
v1018add hl,h1 ;3m 11t    1b      shift rs1t low word
v1019exx      ;1m 4t      1b      main registers
v1020rl l      ;2m 8t      2b      shift rs1t high word
v1021rl h      ;2m 8t      2b
v1022jp $-011h ;3m 10t    3b      check to see if normalized
v1023bit 6,h    ;2m 8t      2b      test ms bit s0.xxxx neg normal
v1024jp z, $+013h ;3m 10t    3b      if 0 in msb then store result
v1025exx      ;1m 4t      1b      prime registers
v1026cp c      ;1m 4t      1b      is exponent zero?
v1027jp z, $+00dh ;3m 10t    3b      if exponent is 0 then stop
v1028dec c      ;1m 4t      1b      decrement exponent in c'
v1029add hl,h1 ;3m 11t    1b      shift rs1t low word
v1030exx      ;1m 4t      1b      main registers
v1031rl l      ;2m 8t      2b      shift rs1t high word
v1032rl h      ;2m 8t      2b
v1033jp $-011h ;3m 10t    3b      check to see if normalized
v1034exx      ;1m 4t      1b      main registers
v1035ld (<rs1t>+1),hl ;5m 16t 3b      save rs1t ms word
v1036exx      ;1m 4t      1b      prime registers
v1037ld (<rs1t>+3),hl ;5m 16t 3b      save rs1t ls word
v1038ld a,c     ;1m 4t      1b      get exponent of rs1t
v1039ld (<rs1t>),a ;4m 13t 3b      save rs1t exponent
v1040exx      ;1m 4t      1b      main registers
v1041endtext
v1042calc romptr=romptr+205
v1043s.subck (rs1t,arg1,arg2:0,8,0,8,0,8,23,87,26,14,0,1043,1057)
v1044com primitive to subtract arg2 from arg1 and store in rs1t
v1045com list= rs1t,arg1,arg2:precisions:s,t,e,c,i,a
v1046begin stext
v1047ld a,<arg1> ;4m 13t 3b      load arg1 in accumulator
v1048ld hl,<arg2> ;3m 10t 3b      point hl to arg2
v1049sub (hl)     ;2m 7t 1b      arg1 - arg2
v1050jp po,$+13 ;3m 10t 3b      if no overflow store result
v1051jp c,$+8     ;3m 10t 3b      if carry the maximize minus rs1t
v1052ld a,01111111b ;2m 7t 2b      put in largest positive value
v1053jp $+5       ;3m 10t 3b
v1054ld a,100000000b ;2m 7t 2b      put in largest negative value
v1055ld (<rs1t>),a ;13t 4m 3b      save result of add in rs1t
v1056endtext
v1057calc romptr=romptr+23
v1058s.exitproc (nam,cont ::6,30,8,11,0,1058,1069)
v1059com primitive to close proc (and reset associated contingency)
v1060com list=proc-nam, contnam:empty; storage, time, ext,calc,incl,addr

```



```

v1061if <cont>.eq.0 skip 4
v1062begin stext
v1063ld a, 0 ;2m 7t 2b reset contingency <cont>
v1064ld (<cont>),a ;4m 13t 3b following completed task
v1065endtext
v1066begin stext
v1067ret ;3m 10t 1b return to monitor,exit <nam>
v1068endtext
v1069calc romptr=romptr+6
v1070s.cons (nam,val, :0,16:2,0,0,6,0,1070,1076)
v1071com primitive to define data for 16 bit integer
v1072com list=data-name,value=value-prec,stor,time,ext,c,i,addrs
v1073begin stext
v1074<nam>: defw <val> ;define a two byte integer
v1075endtext
v1076calc romptr=romptr+2
v1077s.var (name:0,16:0,0,0,3,0,1077,1085)
v1078com primitive to define storage for 16 bit variable integer
v1079com list=data-name,value=value-prec,stor,time,ext,c,i,addrs
v1080calc ramptr=ramptr - 2
v1081begin stext
v1082org <ramptr> ;16 bit variable <name> in ram
v1083<name>: defw 0 ;0m 0t 2b
v1084org <romptr>
v1085endtext
v1086s.ftptieee (rslt,arg1:0,32,0,32,:105,558,140,67,0,1086,1153)
v1087com primitive to convert the floating point to the ieee standard
v1088com format exp,byte2,sign,byte1,byte4,byte3 to sign exp .magnitude
v1089com list=rslt,arg1:precisions:s,t,e,c,i,addr
v1090begin stext
v1091ld a,(<arg1>) ;4m 13t 3b get exponent of arg2
v1092add a, 080h ;2m 7t 2b put in offset
v1093ld c, a ;1m 4t 1b save exponent in c'
v1094ld a, 080h ;2m 7t 2b mask for sign
v1095and d, 6 ;1m 4t 1b get sign of rslt
v1096ld b, 6 ;2m 7t 2b number of shifts
v1097ld de,(<arg1>+1);5m 16t 4b first word
v1098ld hl,(<arg1>+3);5m 16t 3b second word
v1099jp m,$+019h ;3m 10t 3b number is minus
v1100srl d ;2m 8t 2b rotate mantissa 6 places
v1101rr e ;2m 8t 2b to the right
v1102rr h ;2m 8t 2b
v1103rr l ;2m 8t 2b
v1104djnz $,-008h ;3m2 13t8 2b
v1105ld d, c ;1m 4t 1b put exp in front of mantissa
v1106srl d ;2m 8t 2b put exp into leading
v1107rr e ;2m 8t 2b of mantissa
v1108rr h ;2m 8t 2b sign is positive
v1109rr l ;2m 8t 2b
v1110jp $+031h ;3m 10t 3b store rslt
v1111ld a, l ;1m 4t 1b change to sign magnitude
v1112cpl ;1m 4t 1b from 2's complement
v1113ld l, a ;1m 4t 1b

```



```

v1114ld a, h      ;1m      4t      1b
v1115cpl          ;1m      4t      1b
v1116ld h, a      ;1m      4t      1b
v1117ld a, e      ;1m      4t      1b
v1118cpl          ;1m      4t      1b
v1119ld e, a      ;1m      4t      1b
v1120ld a, d      ;1m      4t      1b
v1121cpl          ;1m      4t      1b
v1122ld d, a      ;1m      4t      1b
v1123inc l        ;1m      4t      1b
v1124ld a, h      ;1m      4t      1b
v1125adc a, 0     ;2m      7t      2b
v1126ld h, a      ;1m      4t      1b
v1127ld a, e      ;1m      4t      1b
v1128adc a, 0     ;2m      7t      2b
v1129ld e, a      ;1m      4t      1b
v1130ld a, d      ;1m      4t      1b
v1131adc a, 0     ;2m      7t      2b
v1132ld d, a      ;1m      4t      1b
v1133srl d        ;2m      8t      2b
v1134rr e         ;2m      8t      2b
v1135rr h         ;2m      8t      2b
v1136rr l         ;2m      8t      2b
v1137djnz $-008h  ;3m2     13t8    2b
v1138ld d, c      ;1m      4t      1b
v1139srl d        ;2m      8t      2b
v1140rr e         ;2m      8t      2b
v1141rr h         ;2m      8t      2b
v1142rr l         ;2m      8t      2b
v1143set 7, d     ;2m      8t      2b
v1144ld a, d      ;1m      4t      1b
v1145ld (<rslt>),a ;3m      13t     3b
v1146ld a, e      ;1m      4t      1b
v1147ld (<rslt>+1),a ;3m      13t     3b
v1148ld a, h      ;1m      4t      1b
v1149ld (<rslt>+2),a ;3m      13t     3b
v1150ld a, l      ;1m      4t      1b
v1151ld (<rslt>+3),a ;3m      13t     3b
v1152endtext
v1153calc romptr=romptr+105
v1154s.loc        (loc :1,4,1,6,0,1154,1160)
v1155com primitive to define a label (location)
v1156com list=label-name :empty: storage,time,ext,calc,incl,addr
v1157begin stext
v1158<loc>: nop
v1159endtext
v1160calc romptr=romptr+1
v1161s.end        (:3,10,3,8,10,1161,1171)
v1162com primitive to end software listing and complete implementation
v1163com list=empty:empty:stor,time,ext,calc,incl,addr
v1164begin stext
v1165@k<initlk>:jp espvsr ;3m 10t 3b initialization of hardware is complete
v1166             ; start top of main monitor loop

```

rotate mantissa 6 places
to the right

put exp in front of mantissa
put exp into leading
of mantissa
sign is positive

change sign to negative
save reslut


```

v1167end
v1168endtext
v1169calc romptr=romptr+3
v1170com put in memory needed for implementation in ram and rom
v1171incl h.memory (:)
v1172s.every (nam: 7,34,9,10,0,1172,1182)
v1173com primitive to define dummy function for every-period statement
v1174com list=proc-nam :empty: storage,time,ext,calc,incl,addr
v1175begin stext
v1176;dummy procedure for every-period type contingency
v1177@<nam>: nop ;1m 4t 1b dummy function entry point
v1178 ld a,1 ;2m 7t 2b force function value
v1179 ld (<nam>),a;3m 13t 3b to true value (1)
v1180 ret ;3m 10t 1b return to monitor
v1181endtext
v1182calc romptr=romptr+7
v1183s.fsub (rs1t,arg1,arg2:0,32,0,32,0,32:206,2263,597,126,0,1183,1309)
v1184com primitive to subtract 2 floating point numbers and store rs1t
v1185com form is rs1t le arg1 - arg2
v1186com format exp,byte1,byte2,byte3,byte4
v1187com mantissa is in form s1.xxxx 01.xxxx or 10.xxxx
v1188com s is sign of number xxx is two's complement number
v1189com list=rs1t,arg1:precisions:s,t,e,c,i,addr
v1190begin stext
v1191ld de,(<arg2>+1);6m 20t 4b first word
v1192ld hl,(<arg1>+1);5m 16t 3b first word
v1193exx ;1m 4t 1b prime registers
v1194ld de, (<arg2>+3);6m 20t 4b second word
v1195ld hl, (<arg1>+3);5m 16t 3b second word
v1196ld a, (<arg2>);4m 13t 3b get exponent of arg1
v1197ld b, a ;1m 4t 1b save exponent in b'
v1198ld a, (<arg1>);4m 13t 3b get exponent of arg2
v1199ld c, a ;1m 4t 1b save exponent in c'
v1200exx ;1m 4t 1b main registers
v1201ld a, 10000000b;2m 8t 2b mask for sign
v1202and d ;1m 4t 1b sign of arg1
v1203ld b, a ;1m 4t 1b save sign of arg1 main
v1204ld a, 10000000b;2m 8t 2b mask for sign
v1205and h ;1m 4t 1b sign of arg2
v1206ld c, a ;1m 4t 1b save signof arg2 main
v1207exx ;1m 4t 1b prime registers
v1208ld a, b ;1m 4t 1b put arg2 exponent in a
v1209sub c ;1m 4t 1b subtract exponent arg1
v1210exx ;1m 4t 1b main registers
v1211jp z,$+041h ;3m 10t 3b exponents are equal no shift required
v1212jp m,$+024h ;3m 10t 3b arg1 gt arg2 exponent
v1213cp 31 ;2m 7t 2b max number of shifts
v1214jp m,$+005h ;3m 10t 3b is shift lt 31
v1215ld a, 31 ;2m 7t 2b limit to 31 shifts
v1216sra h ;2m 8t 2b shift msb arg2 right retain sign
v1217rr l ;2m 8t 2b rotate byte2 arg2
v1218exx ;1m 4t 1b prime registers
v1219rr h ;2m 8t 2b rotate byte3 arg2

```


v1220rr	1	:2m	8t	2b	rotate byte4 arg2
v1221jp	nc, \$+005h	:3m	10t	3b	sticky bit
v1222set	0, 1	:2m	8t	2b	set least significant bit
v1223exx		:1m	4t	1b	return to main registers
v1224dec	a	:1m	4t	1b	
v1225jp	nz, \$-010h	:3m	10t	3b	check to see if exp are aligned
v12261d	c, b	:1m	4t	1b	sign of result will be same as arg1
v1227exx		:1m	4t	1b	prime registers
v12281d	c, b	:1m	4t	1b	exponent for result is in c'
v1229exx		:1m	4t	1b	main registers
v1230jp	\$+01dh	:3m	10t	3b	justified continue with add
v1231cp	-31	:2m	7t	2b	is shifts lt -31
v1232jp	p, \$+005h	:3m	10t	3b	is shifts lt 31
v12331d	a, -31	:2m	7t	2b	limit to 31 shifts
v1234sra	d	:2m	8t	2b	shift msb arg1 right keep sign
v1235rr	e	:2m	8t	2b	rotate byte2 arg1
v1236exx		:1m	4t	1b	prime registers
v1237rr	d	:2m	8t	2b	rotate byte3 arg1
v1238rr	e	:2m	8t	2b	rotate byte4 arg1
v1239jp	nc, \$+005h	:3m	10t	3b	sticky bit
v1240set	0, e	:2m	8t	2b	set least significant bit
v1241exx		:1m	4t	1b	main registers
v1242inc	a	:1m	4t	1b	add one to negative number till 0
v1243jp	nz, \$-010h	:3m	10t	3b	prime register begin subtract
v1244exx		:1m	4t	1b	clear carry
v1245and	a	:1m	4t	1b	subtract 3 and 4 bytes of mantissa
v1246sbc	h1, de	:4m	15t	2b	main registers
v1247exx		:1m	4t	1b	prepare to add byte2
v12481d	a, 1	:1m	4t	1b	sub byte2
v1249sbc	a, e	:1m	4t	1b	save byte2
v12501d	1, a	:1m	4t	1b	prepare to add byte1
v12511d	a, h	:1m	4t	1b	sub byte1
v1252sbc	a, d	:1m	4t	1b	save byte1 of result
v12531d	h, a	:1m	4t	1b	correct for overflow
v1254jp	pe, \$+006h	:3m	12t	1b	no overflow
v1255jp	\$+00eh	:3m	10t	3b	shift msb of mantissa of rs1t right
v1256rr	h	:2m	8t	2b	rotate byte2
v1257rr	l	:2m	8t	2b	prime registers
v1258exx		:1m	4t	1b	rotate byte3
v1259rr	h	:2m	8t	2b	rotate byte4
v1260rr	l	:2m	8t	2b	correct exponent in c'
v1261inc	c	:1m	4t	1b	return to main registers
v1262exx		:1m	4t	1b	put zero in accumulator -- zero rs1t?
v12631d	a, 0	:2m	8t	2b	is msb zero
v1264cp	h	:1m	4t	1b	quit test of rs1t if non zero
v1265jr	nz, \$+010h	:2m	7t	2b	is byte2 zero
v1266cp	l	:1m	4t	1b	quit test of rs1t if non zero
v1267jr	nz, \$+00dh	:2m	7t	2b	prime registers
v1268exx		:am	4t	1b	is byte 2 zero
v1269cp	h	:1m	4t	1b	quit test of rs1t and flip reg non zero
v1270jr	nz, \$+008h	:2m	7t	2b	is byte 3 zero
v1271cp	l	:1m	4t	1b	quit test of rs1t and flip reg non zero
v1272jr	nz, \$+005h	:2m	7t	2b	

v1273exx	;1m	4t	1b	main registers arg2 is 0
v1274jr \$+031h	;3m	12t	2b	result is zero store
v1275exx	;1m	4t	1b	main registers
v1276ld a,h	;1m	4t	1b	test sign of result
v1277and a	;1m	4t	1b	set sign bit
v1278jp m,\$+017h	;3m	10t	3b	if neg goto neg normalization
v1279bit 6,h	;2m	8t	2b	test ms bit sl,xxxx pos normal
v1280jp nz,\$+027h	;3m	10t	3b	if 1 in msb then store result
v1281exx	;1m	4t	1b	prime registers
v1282cp c	;1m	4t	1b	is exponent zero?
v1283jp z,\$+021h	;3m	10t	3b	if exponent is 0 then stop
v1284dec c	;1m	4t	1b	decrement exponent in c'
v1285add h1,h1	;3m	11t	1b	shift rs1t low word
v1286exx	;1m	4t	1b	main registers
v1287rl l	;2m	8t	2b	shift rs1t high word
v1288rl h	;2m	8t	2b	
v1289jp \$-011h	;3m	10t	3b	check to see if normalized
v1290bit 6,h	;2m	8t	2b	test ms bit s0.xxxx neg normal
v1291jp z,\$+013h	;3m	10t	3b	if 0 in msb then store result
v1292exx	;1m	4t	1b	prime registers
v1293cp c	;1m	4t	1b	is exponent zero?
v1294jp z,\$+00dh	;3m	10t	3b	if exponent is 0 then stop
v1295dec c	;1m	4t	1b	decrement exponent in c'
v1296add h1,h1	;3m	11t	1b	shift rs1t low word
v1297exx	;1m	4t	1b	main registers
v1298rl l	;2m	8t	2b	shift rs1t high word
v1299rl h	;2m	8t	2b	
v1300jp \$-011h	;3m	10t	3b	check to see if normalized
v1301exx	;1m	4t	1b	main registers
v1302ld (<rs1t>+1),h1;5m	16t	3b		save rs1t ms word
v1303exx	;1m	4t	1b	prime registers
v1304ld (<rs1t>+3),h1;5m	16t	3b		save rs1t ls word
v1305ld a,c	;1m	4t	1b	get exponent of rs1t
v1306ld (<rs1t>).a ;4m	13t	3b		save rs1t exponent
v1307exx	;1m	4t	1b	main registers
v1308endtext				
v1309calc romptr=romptr+206				
v1310s.jmpf (val,loc:0;8;30;8;8,0,1310,1318)				
v1311com primitive to branch on true condition (pollack)(change smith)				
v1312com list=value,location:bits:stor,time,ext,calc,incl,addrs				
v1313begin stext				
v1314ld a,<val>	;3m 13t 3b	branch to <loc> if <val> is true		
v1315cp 0	;2m 7t 2b			
v1316jp nz,<loc>	;3m 10t 3b			
v1317endtext				
v1318calc romptr=romptr+8				
v1319s.jmpf (val,loc :0;8;30;10;8,0,1319,1327)				
v1320com primitive to branch on false condition				
v1321com list=value,jump-loc: value-prec.:storage,time,ext,calc,incl,addr				
v1322begin stext				
v1323ld a,<val>	;3m 13t 3b	branch to <loc> if <val> is true		
v1324cp 0	;2m 7t 2b			
v1325jp z,<loc>	;3m 10t 3b			


```

v1326endtext
v1327calc romptr=romptr+8
v1328s.monitor ( :7,27,7,10,0,1328,1338)
v1329com primitive to define p2 monitor as controller supervisor
v1330com list = empty:empty: storage,time,ext,calc,int,addr
v1331begin stext
v1332:           =monitor section=
v1333@spvsr:ld a,(@initvar);3m 13t 3b mark top of the polling loop and test
v1334           ;           to see if the initializations have been
v1335 and a       ;1m      4t 1b done. if not do so
v1336 jp z,@initial;3m 10t 3b
v1337endtext
v1338calc romptr=romptr+7
v1339s.tabend ( : : 3,10,3,6,0,1339,1345)
v1340com subroutine to define end of monitor table
v1341com list= empty:empty:s,t,e,c,i,addr)
v1342begin stext
v1343jp @spvsr      ;go to the top of the polling loop of monitor table
v1344endtext
v1345calc romptr=romptr+3
v1346s.varfp (name:0,16:0,0,0,3,0,1346,1356)
v1347com primitive to define storage for a floating point number
v1348com list=data-name,value:value-prec,stor,time,ext,c,i,adds
v1349calc romptr=romptr - 5
v1350begin stext
v1351org <romptr>      ;fp variable <name> in ram
v1352<name>: defw 0      ;0m 0t 2b
v1353 defw 0           ; 2b
v1354 defb 0           ; 1b
v1355org <romptr>
v1356endtext
v1357s.fmul (rslt,arg1,arg2:0,32,0,32,0,32:204,2263,597,0,0,1357,1519)
v1358com primitive to multiply two floating point numbers and
v1359com store the result in rslt
v1360com rslt le arg1 * arg2
v1361com format exp,byte1,byte2,byte3,byte4
v1362com mantissa is in form s.xxxxx
v1363com s is sign of number xxx is two's complement number
v1364com list=rslt,arg1:precisions:s,t,e,c,i,addr
v1365begin stext
v1366ld de,(<arg1>+1);6m 20t 4b first word
v1367ld hl,0          ;3m 10t 3b zero hl for result
v1368exx              ;1m 4t 1b prime registers
v1369ld de,(<arg1>+3);6m 20t 4b second word
v1370ld hl,0          ;5m 10t 3b zero hl for result
v1371ld a,(<arg1>)    ;3m 13t 3b get arg1 exponent
v1372ld ix,<arg2>      ;4m 14t 4b put address of arg2 in ix
v1373add a,(ix+0)      ;5m 19t 3b results exponent
v1374jp pe,$+227       ;3m 12t 3b if exponent is out of range fix
v1375ld c,a           ;1m 4t 1b save exponent in c,
v1376exx              ;1m 4t 1b main registers
v1377ld a,(ix+3)       ;5m 19t 3b load byte 4 of mantissa
v1378ld b, 8           ;2m 7t 2b prepare for mult 4 byte

```



```

v1379srl a          ;2m 8t 2b      shift multiplier
v1380jp nc,$+8      ;3m 10t 3b      if no 1 don't add
v1381exx            ;1m 4t 1b      prime register beginning of add
v1382add hl, de     ;3m 11t 1b      add 3 and 4 bytes of mantissa
v1383exx            ;1m 4t 1b      main registers
v1384adc hl, de     ;4m 15t 2b      add bytes 1 and 2
v1385 rr h          ;2m 8t 2b      shift msb of mantissa of rs1t right
v1386rr l           ;2m 8t 2b      rotate byte2
v1387exx            ;1m 4t 1b      prime registers
v1388rr h           ;2m 8t 2b      rotate byte3
v1389rr l           ;2m 8t 2b      rotate byte4
v1390exx            ;1m 4t 1b      return to main registers
v1391djnz $-20      ;3m2 13t8 2b
v1392ld a,(ix+4)    ;5m 19t 3b      load byte 4 of mantissa
v1393ld b, 8        ;2m 7t 2b      prepare for mult 4 byte
v1394srl a          ;2m 8t 2b      shift multiplier
v1395jp nc,$+8      ;3m 10t 3b      if no 1 don't add
v1396exx            ;1m 4t 1b      prime register beginning of add
v1397add hl, de     ;3m 11t 1b      add 3 and 4 bytes of mantissa
v1398exx            ;1m 4t 1b      main registers
v1399adc hl, de     ;4m 15t 2b      add bytes 1 and 2
v1400 rr h          ;2m 8t 2b      shift msb of mantissa of rs1t right
v1401rr l           ;2m 8t 2b      rotate byte2
v1402exx            ;1m 4t 1b      prime registers
v1403rr h           ;2m 8t 2b      rotate byte3
v1404rr l           ;2m 8t 2b      rotate byte4
v1405exx            ;1m 4t 1b      return to main registers
v1406djnz $-20      ;3m2 13t8 2b
v1407ld a,(ix+1)    ;5m 19t 3b      load byte 4 of mantissa
v1408ld b, 8        ;2m 7t 2b      prepare for mult 4 byte
v1409srl a          ;2m 8t 2b      shift multiplier
v1410jp nc,$+8      ;3m 10t 3b      if no 1 don't add
v1411exx            ;1m 4t 1b      prime register beginning of add
v1412add hl, de     ;3m 11t 1b      add 3 and 4 bytes of mantissa
v1413exx            ;1m 4t 1b      main registers
v1414adc hl, de     ;4m 15t 2b      add bytes 1 and 2
v1415 rr h          ;2m 8t 2b      shift msb of mantissa of rs1t right
v1416rr l           ;2m 8t 2b      rotate byte2
v1417exx            ;1m 4t 1b      prime registers
v1418rr h           ;2m 8t 2b      rotate byte3
v1419rr l           ;2m 8t 2b      rotate byte4
v1420exx            ;1m 4t 1b      return to main registers
v1421djnz $-20      ;3m2 13t8 2b
v1422ld a,(ix+4)    ;5m 19t 3b      load byte 4 of mantissa
v1423ld b, 6        ;2m 7t 2b      prepare for mult 4 byte
v1424srl a          ;2m 8t 2b      shift multiplier
v1425jp nc,$+8      ;3m 10t 3b      if no 1 don't add
v1426exx            ;1m 4t 1b      prime register beginning of add
v1427add hl, de     ;3m 11t 1b      add 3 and 4 bytes of mantissa
v1428exx            ;1m 4t 1b      main registers
v1429adc hl, de     ;4m 15t 2b      add bytes 1 and 2
v1430 rr h          ;2m 8t 2b      shift msb of mantissa of rs1t right
v1431rr l           ;2m 8t 2b      rotate byte2

```


v1432exx	:1m 4t 1b	prime registers
v1433rr h	:2m 8t 2b	rotate byte3
v1434rr l	:1m 4t 2b	rotate byte4
v1435exx	:1m 4t 1b	return to main registers
v1436djnz \$-20	:3m2 13t8 2b	
v1437srl a	:2m 8t 2b	shift multiplier
v1438jp nc,\$+8	:3m 10t 3b	if no 1 don't add
v1439exx	:1m 4t 1b	prime register beginning of add
v1440add hl, de	:3m 11t 1b	add 3 and 4 bytes of mantissa
v1441exx	:1m 4t 1b	main registers
v1442adc hl, de	:4m 15t 2b	add bytes 1 and 2
v1443 sra h	:2m 8t 2b	shift msb of mantissa of rslt right
v1444rr l	:2m 8t 2b	rotate byte2
v1445exx	:1m 4t 1b	prime registers
v1446rr h	:2m 8t 2b	rotate byte3
v1447rr l	:2m 8t 2b	rotate byte4
v1448exx	:1m 4t 1b	return to main registers
v1449srl a	:2m 8t 2b	shift multiplier
v1450jp nc,\$+27	:3m 10t 3b	if no 1 don't sub
v1451and a	:1m 4t 1b	clear carry
v1452exx	:1m 4t 1b	prime register beginning of add
v1453sbc hl, de	:4m 15t 2b	correct rslt if sign is neg
v1454exx	:1m 4t 1b	main registers
v1455sbc hl, de	:4m 15t 2b	
v1456jp pe,\$+6	:3m 10t 3b	
v1457jp \$+14	:3m 10t 3b	
v1458 srl h	:2m 8t 2b	shift msb of mantissa of rslt right
v1459rr l	:2m 8t 2b	rotate byte2
v1460exx	:1m 4t 1b	prime registers
v1461rr h	:2m 8t 2b	rotate byte3
v1462rr l	:2m 8t 2b	rotate byte4
v1463inc c	:1m 4t 1b	correct for overflow
v1464exx	:1m 4t 1b	return to main registers
v1465ld a, 0	:2m 8t 2b	put zero in accumulator -- zero rslt?
v1466cp h	:1m 4t 1b	is msb zero
v1467jr nz, \$+17	:2m 7t 2b	quit test of rslt if non zero
v1468cp l	:1m 4t 1b	is byte2 zero
v1469jr nz, \$+14	:2m 7t 2b	quit test of rslt if non zero
v1470exx	:am 4t 1b	prime registers
v1471cp h	:1m 4t 1b	is byte 2 zero
v1472jr nz, \$+9	:2m 7t 2b	quit test of rslt and flip reg non zero
v1473cp l	:1m 4t 1b	is byte 3 zero
v1474jr nz, \$+6	:2m 7t 2b	quit test of rslt and flip reg non zero
v1475ld c, a	:1m 4t 1b	make exponent 0 in c'
v1476exx	:1m 4t 1b	main registers arg2 is 0
v1477jr \$+66	:3m 12t 2b	result is zero store
v1478exx	:1m 4t 1b	main registers
v1479 ld a,h	:1m 4t 1b	test sign of result
v1480and a	:1m 4t 1b	set sign bit
v1481jp m,\$+23	:3m 10t 3b	if neg goto neg normalization
v1482 bit 6, h	:2m 8t 2b	test ms bit sl.xxxx pos normal
v1483jp nz, \$+56	:3m 10t 3b	if 1 in msb then store result
v1484exx	:1m 4t 1b	prime registers


```

v1485cp c 4t 1b is exponent zero?
v1486jp z, $+50 if exponent is 0 then stop
v1487dec c 3m 10t 3b decrement exponent in c,
v1488add hl, hl 3m 4t 1b shift rslt low word
v1489exx 3m 11t 1b main registers
v1490rl 1 1m 4t 1b shift rslt high word
v1491rl h 2m 8t 2b
v1492jp $-17 3m 10t 3b check to see if normalized
v1493 bit 6, h 2m 8t 2b test ms bit s0.xxxx neg normal
v1494jp z, $+36 3m 10t 3b if 0 in msp then store result
v1495exx 1m 4t 1b prime registers
v1496cp c 1m 4t 1b is exponent zero?
v1497jp z, $+30 3m 10t 3b if exponent is 0 then stop
v1498dec c 1m 4t 1b decrement exponent in c,
v1499add hl, hl 3m 11t 1b shift rslt low word
v1500exx 1m 4t 1b main registers
v1501rl 1 2m 8t 2b shift rslt high word
v1502rl h 2m 8t 2b
v1503jp $-17 3m 10t 3b check to see if normalized
v1504 rr a 2m 8t 2b fix sign
v1505ld c,0 2m 7t 2b make underflow 0
v1506jp m,$+5 3m 10t 3b if overflow neg make zero
v1507ld c,01111111b;2m 7t 2b make infinity
v1508 ld hl,0 3m 10t 3b
v1509exx 1m 4t 1b main registers
v1510ld hl,0 1m 4t 3b
v1511exx 1m 4t 1b prime registers
v1512 exx 1m 4t 1b main registers
v1513 ld (<rslt>+1),hl;5m 16t 3b save rslt ms word
v1514exx 1m 4t 1b prime registers
v1515ld (<rslt>+3),hl;5m 16t 3b save rslt ls word
v1516ld a, c 1m 4t 1b get exponent of rslt
v1517ld (<rslt>),a 4m 13t 3b save rslt exponent
v1518exx 1m 4t 1b main registers
v1519endtext
v1520s.atod (signam:0,8:43,641,23,9,11,1520,1610)
v1521com primitive to convert an analog signal to a digital signal
v1522com signam is the name of the signal to be stored
v1523com initlk is a counter indicating the last link to jump to in any
v1524com hardware required initialization.
v1525com natode is the number of 8 bit a to d ports that have been requested
v1526com this is used to allow more than one channel per a to d port.
v1527com this primitive is currently set up to include 2 a to d boards
v1528com for a total of 32 possible channels.
v1529calc natode = natode + 1
v1530if natode .gt. 1 skip 3
v1531incl h.atod (:)
v1532calc natodp = 0
v1533com the first a to d board is ported at address 000<natodp>
v1534begin stext
v1535jp $+01ch ;3b start of initialization for first 8 bit a to d board
v1536w<initlk>;ld b, 3 ;2b number of bytes to output
v1537ld c, <natodp>+1 ;3b atodp = 0 port to be loaded

```



```

v1538ld hl,$+00eh ;3b
v1539otir ;2b
v1540ld b, 2 ;2b number of bytes for output
v1541ld c, <natodp>+3 ;2b atodp = 0 port to be loaded
v1542otir ;2b
v1543jp $+008h ;3b
v1544defb 0cfh ;1b
v1545defb 080h ;1b
v1546defb 007h ;1b
v1547defb 04fh ;1b
v1548defb 007h ;1b
v1549endtext
v1550calc initlk=initlk+1
v1551begin stext
v1552jp @i<initlk> ; 3b jump to next hardware initialization
v1553 nop ;1b end of initialization for first 8 bit a to d board
v1554endtext
v1555if natode .lt. 16 skip 4
v1556if natode .gt. 16 skip 3
v1557incl h.atod (:)
v1558calc natodp = 4
v1559com the second atod board is ported at address 000<natodp>
v1560begin stext
v1561jp $+01ch ;3b start of initialization for first 8 bit a to d board
v1562@i<initlk>:ld b, 3 ;2b number of bytes to output
v1563ld c, <natodp>+1 ;2b atodp = 4 port to be loaded
v1564ld hl,$+00ch ;3b
v1565otir ;2b
v1566ld b, 2 ;2b number of bytes for output
v1567ld c, <natodp>+3 ;2b atodp = 4 port to be loaded
v1568otir ;2b
v1569jp $+008h ;3b
v1570defb 0cfh ;1b
v1571defb 080h ;1b
v1572defb 007h ;1b
v1573defb 04fh ;1b
v1574defb 007h ;1b
v1575endtext
v1576calc initlk=initlk+1
v1577begin stext
v1578jp @i<initlk> ;3b
v1579 nop ;1b end of initialization for second 8 bit a to d board
v1580endtext
v1581com list=source: input-lines, , conds, ,: storage,time,ext,calc,incl,addr
v1582if natode .lt. 16 skip 1
v1583calc scrch = natode - 16
v1584if debug .eq. 0 skip 11
v1585begin stext
v1586ld de, $+0013h
v1587ld c, 9
v1588call <bdos>
v1589ld c, 1 ;bdos call requesting a character from the console
v1590call <bdos>

```



```

v1591ld (<signam>), a ;save the results of the input
v1592jp $+021h
v1593 defm "requesting input for <signam>$"
v1594 nop ;keep message out of straight line execution
v1595endtext
v1596if debug .eq. 1 skip 13
v1597begin stext
v1598ld a,<scrch> ;3m 13t 3b channel to be selected for input
v1599out(<natodp>),a ;3m 11t 2b clear control
v1600or 060h ;1m 4t 1b set start conv, addr latch
v1601out(<natodp>),a ;3m 11t 2b issue a/d control
v1602in a,(<natodp>) ;3m 11t 2b read status
v1603bit 7, a ;2m 8t 2b check done bit
v1604jr z, $-4 ;2m 7t 2b loop till done
v1605conversion time is 138 microseconds + one full execution of
v1606done polling loop
v1607in a,(<natodp>+2);3m 11t 2b read a/d data
v1608ld (<signam>),a ;3m 13t 3b save results of input in <signam>
v1609endtext
v1610calc romptr = romptr + 43
v1611s.blockexit (switch:0,8:5,20,5,8,0,1611,1619)
v1612com primitive to set the switch associated with a submonitor to
v1613com false and thus cause the submonitor to exit to the main
v1614com monitor at the bottom of the submonitor loop
v1615begin stext
v1616ld a, 0 ;2m 7t 2b
v1617ld (<switch>),a ;3m 13t 3b set switch to false
v1618endtext
v1619calc romptr=romptr+5
v1620s.blockcons (submon:0,255:1,4,1,9,0,1620,1629)
v1621com primitive to mark the beginning of a submonitor. it is used
v1622com along with the blockend to mark the end of a block to mark the
v1623com end of a submonitor block and the blockexit to orderly leave
v1624com block after all code is tested in the block.
v1625com blockstart will test and jump out of main monitor to submonitor
v1626begin stext
v1627<submon>: nop ;1m 4t 1b mark top of submonitor
v1628endtext
v1629calc romptr=romptr+1
v1630s.blockstart(switch,submon:0,8,0,255:7,27,7,9,0,1630,1639)
v1631com primitive to cause the variable switching on the submonitor to
v1632com be tested and if true the submonitor to be executed
v1633begin stext
v1634ld a, (<switch>) ;3m 13t 3b get switch value
v1635and a ;1m 4t 1b set flags
v1636jp nz <submon> ;3m 10t 3b if true execute the
v1637 ; the submonitor
v1638endtext
v1639calc romptr=romptr+7
v1640s.mult (rs1t,arg1,arg2:0,16,0,16,0,16:39,1105,289,22,0,1640,1662)
v1641com multiply 2 16 bit numbers and get 16 bit result
v1642begin stext
v1643ld de,(<arg1>);6m 20t 4b put arg1 in de

```



```

v1644ld bc,(<arg2>);6m      20t      4b      load arg2
v1645ld a, b ;1m            4t 1b      split arg2 to aandc
v1646ld hl, 0 ;3m          10t 3b      clear rslt
v1647ld b, 15d ;2m         7t 2b      set counter to 7bits
v1648rra ;1m            4t 1b
v1649rr c ;2m          8t 2b
v1650jp nc, $+4 ;3m        10t 3b
v1651add hl, de ;3m        11t 1b
v1652sla e ;2m          8t 2b
v1653rl d ;2m          8t 2b
v1654djnz $-00bh ;3m       13t 2b *7 +2m 8t on last time
v1655rra ;1m            4t 1b
v1656rr c ;2m          8t 2b
v1657jp nc, $+6 ;3m        10t 3b
v1658and a ;1m          4t 1b
v1659sbc hl, de ;4m        15t 2b
v1660ld (<rslt>),hl;5m      16t 3b      save result
v1661endtext
v1662calc romptr=romptr+39
v1663s.out (signam,direct:0,8:14,45,12,6,13,1663,1687)
v1664com primitive to allow binary output on any of 6 channels
v1665com signam is the name of an 8 bit variable for output
v1666com ninout is the number of 8 bit in/out channels that have been
v1667com requested. this primitive is current setup to allow only
v1668com one input/output board.
v1669calc ninout = ninout + 1
v1670if ninout .le. 6 skip 4
v1671begin stext
v1672you have requested more than 6 in/out channels on the mdx-diobl
v1673board. currently the primitive is set up to light only 6 lights
v1674endtext
v1675if ninout .gt. 1 skip 1
v1676incl h.inout (:)
v1677calc scrctch= 2+*(ninout-1)+128
v1678begin stext
v1679ld a, (<signam>) ;3m 13t 3b find out if switch is on
v1680cp 1 ;2m 7t 2b
v1681jp nz, @@<arnd> ;3m 10t 3b if switch is off don't output address
v1682@l<arnd>:out (<scrctch>),a ;3m 11t 2b light the lamp
v1683jp @l<arnd> ;3m 10t 3b keep light on forever
v1684@@<arnd>:nop ;1m 4t 1b don't light the lamp
v1685endtext
v1686calc arnd=arnd+1
v1687calc romptr = romptr + 14
v1688s.mult (rslt,arg1,arg2:0,8,0,8,0,16:33,521,137,21,0,1688,1709)
v1689com multiply 2 8 bit number and get 16 bit result
v1690begin stext
v1691ld a,(<arg1>);3m      13t 3b      put arg1 in e
v1692ld e, a, ;1m          4t 1b
v1693ld a,(<arg2>);2m      7t 2b      load arg2
v1694ld hl, 0 ;3m          10t 3b      clear rslt
v1695ld d, h ;1m          4t 1b      clear d for shifts
v1696ld b, 7 ;2m          7t 2b      set counter to 7bits

```



```

v1697rra      :1m      4t      1b
v1698jp      nc,$+4    :3m      10t     3b
v1699add hl, de      :3m      11t     1b
v1700sla e      :2m      8t      2b
v1701rl d      :2m      8t      2b
v1702djnz $-9    :3m      13t     2b *7 +2m 8t on last time
v1703rra      :1m      4t      1b
v1704jp      nc, $+6   :3m      10t     3b
v1705and a      :1m      4t      1b
v1706sbc hl, de    :4m      15t     2b
v1707ld (<rslt>),hl:5m     16t     3b
v1708endtext
v1709calc romptr=romptr+33
v1710s.mult    (rslt,arg1,arg2:0,8,0,8,0,8:34,522,137,22,0,0,1710,1732)
v1711com binary multiplication primitive
v1712begin stext
v1713ld a,(<arg1>):3m      13t     3b      put arg1 in e
v1714ld e, a      :1m      4t      1b
v1715ld a,(<arg2>):2m      7t      2b      load arg2
v1716ld hl, 0      :3m      10t     3b      clear rslt
v1717ld d, h      :1m      4t      1b      clear d for shifts
v1718ld b, 7      :2m      7t      2b      set counter to 7bits
v1719rra      :1m      4t      1b
v1720jp      nc,$+4    :3m      10t     3b
v1721add hl, de    :3m      11t     1b
v1722sla e      :2m      8t      2b
v1723rl d      :2m      8t      2b
v1724djnz $-9    :3m      13t     2b *7 +2m 8t on last time
v1725rra      :1m      4t      1b
v1726jp      nc, $+6   :3m      10t     3b
v1727and a      :1m      4t      1b
v1728sbc hl, de    :4m      15t     2b
v1729ld a, 1      :1m      4t      1b
v1730ld (<rslt>),a :4m      13t     3b      truncate result to 8 bits
v1731endtext      save result
v1732calc romptr=romptr+34
v1733s.blockend (switch,submon:0,8,0,255:10,37,10,9,0,1733,1742)
v1734com primitive to mark the end of a submonitor. it will test switch
v1735com if switch is false it will return to the supervisor.
v1736begin stext
v1737ld a, (<switch>) :3m      13t     3b      get switch
v1738and a      :1m      4t      1b      set flags
v1739jp z, @spvsr :3m      10t     3b      if false goto main monitor
v1740jp <submon> :3m      10t     3b      else jump to top of submonitor
v1741endtext
v1742calc romptr=romptr+10
v1743s.warm      (:3,10,3,6,0,1743,1749)
v1744com primitive to cause the system to warm boot. that is to reinitialize
v1745com the stack and restart the main monitor loop.
v1746begin stext
v1747jp @spvsr :3m      10t     3b
v1748endtext
v1749calc romptr=romptr+3

```



```

v1750s.ifcons      (arg1,ifmark:0,8,0,255:7,27,7,9,0,1750,1759)
v1751com primitive to generate an if-then statement. it is used with
v1752com the construction ifend to mark the end of the executable
v1753com statements.
v1754begin stext
v1755ld a,(<arg1>) ;3m 13t 3b get result of logical expression
v1756and a, ;1m 4t 1b
v1757jp z, <ifmark> ;3m 10t 3b
v1758endtext
v1759calc romptr=romptr+7
v1760s.cold (.:3,10,3,7,0,1760,1767)
v1761com primitive to cause the system to cold boot. that is to reinitialize
v1762com all i/o ports and all variables that have a hard assignment as
v1763com well as resetting the stack.
v1764begin stext
v1765jp @cold ;3m 10t 3b
v1766endtext
v1767calc romptr=romptr+3
v1768s.dtoa (signam:0,8:43,641,23,10,12,1768,1825)
v1769com primitive to convert a digital signal to an analog signal
v1770com signam is the name of the signal to be converted
v1771com initlk is a counter indicating the last link to jump to in any
v1772com hardware required initialization.
v1773com ndtoae is the number of 8 bit d to a ports that have been requested
v1774com this is used to allow more than one channel per d to a port.
v1775com this primitive is currently set up to include 2 d to a boards
v1776com for a total of 32 possible channels.
v1777com @earnd is a scratch link for miscellaneous jumps
v1778calc ndtoae = ndtoae + 1
v1779if ndtoae .gt. 1 skip 3
v1780incl h.dtoa (.:)
v1781calc dtoap = 0
v1782com the first a to d board is ported at address 000<dtoap>
v1783begin stext
v1784jp around ;3b start of initialization for first 8 bit a to d board
v1785@i<initlk>:nop ;2b number of bytes to output
v1786endtext
v1787calc initlk=initlk+1
v1788begin stext
v1789jp @i<initlk> ; 3b jump to next hardware initialization
v1790around: nop ;1b end of initialization for first 8 bit a to d board
v1791endtext
v1792if ndtoae .lt. 16 skip 4
v1793if ndtoae .gt. 16 skip 3
v1794incl h.dtoa (.:)
v1795calc dtoap = 4
v1796com the second dtoa board is ported at address 000<dtoap>
v1797begin stext
v1798jp around2 ;3b start of initialization for second 8 bit a to d board
v1799@i<initlk>:nop ;2b number of bytes to output
v1800endtext
v1801calc initlk=initlk+1
v1802begin stext

```



```

v1803jp @i<initlk> ; 3b jump to next hardware initialization
v1804around2: nop ;1b end of initialization for first 8 bit a to d board
v1805endtext
v1806if ndtoae .lt. 16 skip 1
v1807calc scrтч = ndtoae - 16
v1808if debug .eq. 0 skip 11
v1809begin stext
v1810ld de, @@<arnd>
v1811ld c, 9
v1812call <bdos>
v1813endtext
v1814ld a, (<signam>)
v1815call @wrtbin
v1816jp @@<arnd>
v1817 defm "output from d to a channel <signam> is$"
v1818@@<arnd>: nop ;keep message out of straight line execution
v1819endtext
v1820calc arnd =arnd+1
v1821if debug .eq. 1 skip 13
v1822begin stext
v1823:output for board
v1824endtext
v1825calc romptr = romptr + 4
v1826s.clockcons (:43,14,4,7,0,1826,1870)
v1827com primitive to generate a 1 millisecond clock. the value of
v1828com current time is stored in @time. time will be stored as a
v1829com 16 bit positive integer variable.
v1830com channel 0 will be used for the clock at factory installed
v1831com address of f0
v1832com interrupt has an overhead of 32m 113t every millisecond
v1833calc ramptr=ramptr-2
v1834begin stext
v1835org 1038
v1836push af
v1837push hl
v1838ld a, 0
v1839out (0f4h),a
v1840ld a, 2
v1841out (0f4h),a
v1842ld hl,@time
v1843inc (hl)
v1844pop hl
v1845pop af
v1846reti
v1847org <ramptr>
v1848@time:defw 0 ;set beginning of time to 0
v1849org <romptr>
v1850jp @@<arnd> ;3m 10t 3b
v1851@i<initlk>;
v1852endtext
v1853calc initlk=initlk+1
v1854begin stext
v1855di ;1m 4t 1b protect system from interrupts

```



```

v1856im 1 ;2m 8t 2b set interrupt to mode 1
v1857ld a,00110100b;2m 7t 1b counter 0 + load lsb then mb+ mode2+ bcd
v1858out (0f3h),a ;3m 11t 2b set mode control
v1859ld a,00 ;2m 7t 1b lsb of 2000 bcd
v1860out (0f0h),a ;3m 11t 2b load counter time channel 0
v1861ld a,20h ;2m 7t 1b msb of 2000 bcd
v1862out (0f0h),a ;3m 11t 2b load counter time channel 0
v1863ld a,0000010b;2m 7t 1b set interrupt on channel 0
v1864out (0f4h),a ;3m 11t 2b
v1865ei ;1m 4t 1b enable interrupts
v1866jp @i<initlk> ;3m 10t 3b
v1867@<arnd>;nop ;1m 4t 3b isolate the initialization
v1868endtext
v1869calc arnd=arnd+1
v1870calc romptr=romptr+43
v1871s.ifend (ifmark:0,255:1,4,1,7,0,1871,1878)
v1872com primitive to mark the end of the condition to be executed in
v1873com an if-then statement.
v1874begin text
v1875<ifmark>;nop ;1m 4t 1b end of statements to be executed
v1876 ; if condition was true
v1877endtext
v1878calc romptr=romptr+1
v1879s.initialcons(:1,4,1,5,0,1879,1884)
v1880com primitive to mark the beginning of the things to be initialized.
v1881begin text
v1882@initial: nop ;1m 4t 1b mark top of initialization
v1883endtext
v1884calc romptr=romptr+1
v1885s.initialend (:8,30,8,9,0,1885,1894)
v1886com primitive to mark the end of the things to be initialized.
v1887com also set initvar flag to true so that initialization is done
v1888com only once.
v1889begin text
v1890ld a, 1 ;2m 7t 2b
v1891ld (@initvar),a;3m 13t 3b set initvar to true
v1892jp @spvr ;3m 10t 3b execute main monitor
v1893endtext
v1894calc romptr=romptr+8
v1895s.whend (whend,whstop:0,255:3,10,3,7,0,1895,1902)
v1896com primitive to generate the end in a while do statement.
v1897com it marks the end of the statement that are to be executed
v1898com if the condition is true.
v1899begin text
v1900<whend>;jp <whstop> ;3m 10t 3b
v1901endtext
v1902calc romptr=romptr+3
v1903s.messout (arg1:0,16:.,.,0,0,1903,1917)
v1904com primitive to print a message to the screen when the debug switch
v1905com is set to true
v1906begin text
v1907@messout:ld de, string
v1908ld c, 9

```



```

v1909call <bdos>
v1910ld de, arg1
v1911call <bdos>
v1912ld de string1
v1913call <bdos>
v1914ret ; and return to the calling program
v1915string: defm "output for $"
v1916string1: defm " = $"
v1917endtext
v1918s.whilecon (arg1,whend,whetop:0,8,0,255:7,27,7,9,0,1918,1927)
v1919com primitive to generate an while do statement. it is used with
v1920com the construction wend to mark the end of the executable
v1921com statements.
v1922begin stext
v1923<whetop>:ld a,(arg1);3m 13t 3b get result of logical expression
v1924and a ;1m 4t 1b
v1925jp z, <whend>+3;3m 10t 3b
v1926endtext
v1927calc romptr=romptr+7
v1928s.wrtbin (:...0,0,1928,1978)
v1929com primitive to write a the contents of a memory location to the
v1930com terminal using a standard 05h call. this routine is used and
v1931com included in code only when the debug switch is set to true.
v1932com the value to be outputed is in reg a. no time constraints are
v1933com given for this primitive since it is used only for debug purposes.
v1934com the code has been copied from a class handout in cs3201 taught
v1935com by a. ross
v1936begin stext
v1937<wrtbin>:push bc ;to save the registers, they are pushed onto
v1938 push de ; the stack
v1939 push hl
v1940 push af
v1941 ld h,8 ;the h reg will count the number of chars
v1942 ld l,00000001b ;the l reg will hold a mask to select
v1943 ; the bit to be transmitted
v1944 ld b,a ;save the number to be transmitted in the b reg
v1945 ld a,l ;get the bit select mask
v1946 rrca ;rotate the mask right one position to the next
v1947 ld l,a ;bit and then save a copy back in the l reg.
v1948 and b ;and the mask and the number together, to set
v1949 ; the zero flag to represent the bit to be sent
v1950 ld e,'0' ;prepare to send the character '0',
v1951 jp z,$+5 ;if the bit was zero, send '0',
v1952 ld e,'1' ;if the jump test fails, change the char to '1'
v1953 ;note: we set the flags with the and inst.
v1954 ; and tested with the jp z inst. the ld does
v1955 ; not affect flags.
v1956 ld c,2 ;any call to the cp/m bdos expects a function
v1957 ; number in reg c. function 2 will write the
v1958 ; code in the e reg to the terminal.
v1959 push hl ;unfortunately, cp/m will erase the cpu registers,
v1960 push bc ; so the ones we care about must be saved
v1961 call <bdos> ;this is the call to the cp/m routine

```



```

v1962      pop      bc      ;restore the registers
v1963      pop      hl
v1964      dec      h
v1965      jp      nz,$-15h; if we have not reached zero, go back to
v1966      ; send another bit of the number.
v1967      ld      e,0dh
v1968      ld      c,2
v1969      call    <bdos>
v1970      ld      e,0ah
v1971      ld      c,2
v1972      call    <bdos>
v1973      pop      af
v1974      pop      hl
v1975      pop      de
v1976      pop      bc
v1977      ret
v1978      endtext
v1979s.ne
v1980com primitive to perform comparison between 2 16-bit numbers
v1981com list=result,argument 1, argument 2 ::stor,time,ext,c,i,adrs
v1982begin stext
v1983ld de,<arg1> ;6m 20t 4b if arg1 arg2 then rslt=ffh de=<arg1>
v1984ld hl,<arg2> ;5m 16t 3b hl=<arg2>
v1985sbc hl,de ;4m 15t 2b
v1986ld a,0 ;2m 8t 1b
v1987jr z,$+3 ;3m 13t 2b result equal
v1988cpl ;1m 4t 1b result not equal
v1989 ld (<rslt>),a ;3m 13t 3b
v1990endtext
v1991calc romptr=romptr+16
v1992s.start (:;3,10,3,6,0,1992,1998)
v1993com 8080 library required a primitive to start execution
v1994com this is included for compatibility but is not required
v1995begin stext
v1996jp @spvsc ;3m 10t 3b jump to top of main monitor loop
v1997endtext
v1998calc romptr=romptr+3
v1999h.inout (:;552,30,15,5,6,1999,2023)
v2000com primitive to include a mostek mdx-D1081 board
v2001com 8 bit 64 channel digital input/output board
v2002com ninout is the number of 8 bit ports that have been requested
v2003com board is set up for use with the a 4 mhz system
v2004calc slot = slot + 1
v2005incl h.tcardcage (:;)
v2006begin htext
v2007 put mostek mdx-d1081 board in slot <slot>
v2008 connect the following jumper pins
v2009 1-2 on j3 a15 = 0
v2010 3-4 on j3 a14 = 0
v2011 5-6 on j3 a13 = 0
v2012 7-8 on j3 a12 = 0
v2013 9-10 on j3 a11 = 0
v2014 11-12 on j3 a10 = 0

```



```

v2015      13-14 on j3      a9 = 0
v2016      15-16 on j3      a8 = 0
v2017      17-18 on j3      a7 = 0
v2018      5-6 on j4 multiple dmas not used
v2019      disconnect the following jumper pins
v2020      19-20 on j3 a6 = 1 set high
v2021      3-4 on j4 limits signal length to 25 feet no wait states
v2022      address space 0000000001xxxxxx
v2023endtext
v2024h.atod      (:;552,30,15,7,8,2024,2064)
v2025com primitive to include a mostek mdx-a/d8 board
v2026com analog to digital conversion module
v2027com natode is the number of 8 bit a to d ports that have been requested
v2028com board is set up for use with the a 4 mhz system
v2029com set jumper j3-1 to j3-2 for 4 mhz clock
v2030if natode .ne. 1 skip 13
v2031calc slot = slot + 1
v2032incl h.tcardcage (:;)
v2033begin htext
v2034      put first mostek mdx-a/d8 board in slot <slot>
v2035      connect the following j3 jumper pin
v2036      1-2
v2037      disconnect the following jumper pins
v2038      j6
v2039      j7
v2040      j4
v2041      connect the following j5 jumper pins
v2042      1-2      a7 = 0
v2043      3-4      a6 = 0
v2044      5-6      a5 = 0
v2045      7-8      a4 = 0
v2046      9-10     a3 = 0
v2047      11-12    a2 = 0
v2048      address space 00000000
v2049endtext
v2050if natode .ne. 17 skip 14
v2051calc slot = slot + 1
v2052incl h.tcardcage (:;)
v2053begin htext
v2054      put second mostek mdx-a/d8 board in slot <slot>
v2055      connect the following j5 jumper pins
v2056      1-2      a7 = 0
v2057      3-4      a6 = 0
v2058      5-6      a5 = 0
v2059      7-8      a4 = 0
v2060      9-10     a3 = 0
v2061      remove jumper on
v2062      11-12    a2 = 1
v2063      address space 00000100
v2064endtext
v2065h.processor (:;...2,3,2065,2089)
v2066com primitive to include z-80 cpu board 4 mhz
v2067calc slot = slot + 1

```



```

v2068incl h.tcardcage (:)
v2069begin htext
v2070 put z-80 cpu board in slot <slot>
v2071 memex high
v2072 set jumpers in the following pattern
v2073 jumper pattern
v2074 w2 010
v2075 w3 001
v2076 w4 010
v2077 w5 1
v2078 w6 001
v2079 w7 01
v2080 w8 110
v2081 w9 1111
v2082 w10 1
v2083 w12 101010
v2084 w13 10
v2085 w14 10
v2086 w15 01
v2087 note numbering is from left to right and from top to bottom.
v2088 address space 0000-7fff
v2089endtext
v2090h.tcardcage (...,0,0,2090,2094)
v2091com primitive to include card cage and power supply for controller
v2092begin htext
v2093 connect powersupply to card cage
v2094endtext
v2095h.memory (...,2,3,2095,2115)
v2096com primitive to include required memory
v2097calc slot = slot + 1
v2098incl h.tcardcage (:)
v2099if romptr.lt. ramptr skip 5
v2100begin htext
v2101 the program space and the variable space have colided
v2102 you do not have enough memory to execute your program
v2103 your memory is limited to 16k
v2104endtext
v2105begin htext
v2106 put 16k memory board in slot <slot>
v2107 set jumpers in the following pattern
v2108 jumper pattern
v2109 w1 1111111
v2110 w2 10
v2111 w3 0
v2112 w4 01
v2113 w5 1
v2114 address range for card is 4000-7fff
v2115endtext
v2116h.tcardcage (...,0,0,2116,2122)
v2117com primitive to limit the number of slots in card cage to 8
v2118if slot.le. 8 skip 4
v2119begin htext
v2120 you have exceeded the maximum number of allowable slots in the

```



```

v2121      card cage. it is limited to 8.
v2122endtext
v2123h.dtoa      (.:552,30,15,4,5,2123,2137)
v2124com digital to analog conversion module
v2125com ndtoa is the number of 8 bit d to a ports that have been requested
v2126if ndtoa .ne. 1 skip 5
v2127calc slot = slot + 1
v2128incl h.tcardcage (.:)
v2129begin htext
v2130      put first dtoa board in slot <slot>
v2131endtext
v2132if ndtoa .ne 17 skip 5
v2133calc slot = slot + 1
v2134incl h.tcardcage (.:)
v2135begin htext
v2136      put second second dtoa board in slot <slot>
v2137endtext
v2138s.div      (rslt,arg1,arg2:0,8,0,8,0,8,56,504,129,41,0,2138,2179)
v2139com routine to divide arg1 by arg2 and store in rslt
v2140com taken from zaks p 137
v2141begin stext
v2142ld a, (<arg1>) ;m4      t13      b3      get dividend
v2143and a      ;1m      4t      1b
v2144ld h,0      ;2m      7t      2b
v2145jp p, $+7      ;3m      10t      3b
v2146cpl      ;1m      4t      1b
v2147inc a      ;1m      4t      1b
v2148ld h,080h      ;2m      7t      2b
v2149ld e, a      ;m1      t4      b1
v2150ld a, (<arg2>) ;m4      t13      b3      get divisor
v2151and a      ;1m      4t      1b
v2152jp p, $+0bh      ;3m      10t      3b
v2153cpl      ;1m      4t      1b
v2154inc a      ;1m      4t      1b
v2155ld c, a      ;1m      4t      1b
v2156ld a, 080h      ;2m      7t      2b
v2157xor h      ;1m      4t      1b
v2158ld h, a      ;1m      4t      1b
v2159ld a, c      ;1m      4t      1b
v2160ld c, a      ;m1      t4      b1
v2161xor a      ;m1      t4      b1
v2162ld b, 8      ;m2      t7      b2
v2163rl e      ;m2      t8      b2
v2164rla      ;m1      t4      b1
v2165sub c      ;m1      t4      b1
v2166jr nc, $+3      ;m3      t12      b2
v2167add a, c      ;m1      t4      b1
v2168djnz $-7      ;m3      t13      b2
v2169ld b, a      ;m1      t4      b1
v2170ld a, e      ;m1      t4      b1
v2171rla      ;m1      t4      b1
v2172cpl      ;m1      t4      b1
v2173bit 7, h      ;2m      8t      2b

```

clear accumulator
set loop counter
rotate

trial subtract
subtract ok
restore accum, set cy
m2 t8 on last loop
put remainder in b
get quotient
shift in last result bit
complement bits


```

v2174jp z,$+5 ;3m 10t 3b
v2175cpl ;1m 4t 1b
v2176inc a ;1m 4t 1b
v2177ld (<rslt>),a ;m4 t13 b3 store quotient in rslt
v2178endtext
v2179calc romptr=romptr+56
v2180s.div (rslt,arg1,arg2:0,16,0,16,0,16:78,1458,374,57,0,2180,2237)
v2181com primitive to divide arg1 by arg2 and store in rslt
v2182com list=rslt,arg1,arg2:precisions:s,t,e,c,i,addr
v2183begin stext
v2184ld hl, (<arg1>);5m 16t 3b load arg1 in hl pair
v2185bit 7, h ;2m 8t 2b
v2186ld b,0 ;2m 7t 2b
v2187jp z, $+12 ;3m 10t 3b
v2188ld a, h ;1m 4t 1b
v2189cpl ;1m 4t 1b
v2190ld h, a ;1m 4t 1b
v2191ld a, l ;1m 4t 1b
v2192cpl ;1m 4t 1b
v2193ld l, a ;1m 4t 1b
v2194inc hl ;1m 6t 1b
v2195ld b, 080h ;1m 4t 1b
v2196ld de, (<arg2>);6m 20t 4b load arg2 in bc pair
v2197bit 7, d ;2m 8t 2b
v2198ld a, 0 ;2m 7t 2b
v2199jp z, $+12 ;3m 10t 3b
v2200ld a, d ;1m 4t 1b
v2201cpl ;1m 4t 1b
v2202ld d, a ;1m 4t 1b
v2203ld a, e ;1m 4t 1b
v2204cpl ;1m 4t 1b
v2205ld e, a ;1m 4t 1b
v2206inc de ;1m 6t 1b
v2207ld a, 080h ;2m 7t 2b
v2208xor b ;1m 4t 1b
v2209ex af,af' ;1m 4t 1b
v2210ld c,l ;1m 4t 1b
v2211ld a,h ;1m 4t 1b
v2212ld b, 16d ;2m 7t 2b
v2213ld hl, 0 ;3m 10t 3b
v2214rl c ;2m 8t 2b loop
v2215rla ;1m 4t 1b
v2216adc hl, hl ;3m 11t 1b
v2217sbc hl,de ;4m 15t 2b
v2218jr nc, $+3 ;3m 12t 2b
v2219add hl, de ;3m 11t 1b sub was ok
v2220ccf ;1m 4t 1b restore accumulator
v2221djnz $-11 ;3m 13t 2b calc result bit
v2222rl c ;2m 8t 2m 8t on =0
v2223rla ;1m 4t 1b
v2224ld h, a ;1m 4t 1b
v2225ld l, c ;1m 4t 1b
v2226ex af,af' ;1m 4t 1b restore sign of rslt

```



```

v2227jp p, $+10      :3m      10t      3b
v2228ld a, h          :1m      4t      1b
v2229cpl              :1m      4t      1b
v2230ld h, a          :1m      4t      1b
v2231ld a, l          :1m      4t      1b
v2232cpl              :1m      4t      1b
v2233ld l, a          :1m      4t      1b
v2234inc hl           :1m      6t      1b
v2235ld (<rs!t>),hl;5m 16t      3b      save result
v2236endtext
v2237calc romptr=romptr+78
v2238h.clock          (:,:,0,0,2238,2243)
v2239com primitive to create an clock in the ctc chip of the z80 cpu board
v2240begin htext
v2241 connect j1-20 to j1-12      this connects channel 0 output to channel 1
v2242
v2243endtext
v2244h.keydisplay(:,:,6,8,2244,2263)
v2245com primitive to add the 7303 keyboard/display card. this primitive
v2246com is called by outled, inkey or outalpha. since the primitive has
v2247com the capability to do all three functions with accompanying software
v2248com the software is separated and the card will be included only once
v2249if keybrd .eq. 1 skip 14
v2250calc keybrd = 1
v2251calc slot = slot + 1
v2252incl h.tcardcage (:,:)
v2253begin htext
v2254 put first prolog std 7303 keyboard/display card in slot <slot>
v2255 connect the following jumper pins
v2256 x6
v2257 y4
v2258 z0
v2259 z1
v2260 disconnect the following jumper pins
v2261 all others
v2262 address space 11000000, 11000001
v2263endtext
v2264h.uart           (:,:,0,0,2264,2278)
v2265begin htext
v2266 this is a dummy primitive to remind you to put in the dual uart card
v2267 if you wish to use the nps loading rom. the require setting are as
v2268 follows.
v2269 set jumpers in the following pattern
v2270 jumper pattern
v2271 w1 01
v2272 w2 01
v2273 w3 10
v2274 sx 0001
v2275 sy 00001000
v2276 address space e0 thru e7
v2277endtext
v2278com this has to be the last line stupid or the machine will bomb!

```


LIST OF REFERENCES

1. "JDR Microdevices," Byte, V. 9, N. 2, February 1984.
2. "1983 DP Salaries the Key Word is Perks," Datamation, September 1983.
3. Manwaring, M., L., "A Computer-Aided Approach to the Design of Digital System Controllers Using Microprocessors," Twelfth Asilomar Conference on Circuits, Systems and Computers, IEEE Computer Society, November 6, 1978.
4. Ross, A. A., Computer Aided Design of Microprocessor-Based Controllers, Ph.D. Thesis, University of California, Davis, 1978.
5. Matelan, M. N., The Automatic Design of Real Time Control Systems, Lawrence Livermore Laboratory, December 10, 1976.
6. Ross, A. A., Computer Aided Design of Microprocessor-Based Controllers, Ph.D. Thesis, University of California, Davis, 1978.
7. Ibid.
8. Pollock, G. G., Further Development and Investigation of Computer-Aided Design of Microprocessor Systems, M.S. Thesis, University of California Davis, December 1981.
9. Manwaring, M., L., "A Computer-Aided Approach to the Design of Digital System Controllers Using Microprocessors," Twelfth Asilomar Conference on Circuits, Systems and Computers, IEEE Computer Society, November 6, 1978.
10. Biehl, G., "Automatic Generation of Optimal Microcomputer Programs for Software-Driven Digital Controllers," Symposium on Microcomputers and Microprocessor Applications, Budapest, October, 1979.
11. Sherlock, B. J., User-Friendly, Syntax Directed Input to a Computer Aided Design System, M.S. Thesis, Naval Postgraduate School, June 1983.
12. Chu, Y., "Concepts of a Microcomputer Design Language," 16th Design Automation Conference Proceedings, IEEE Computer Society, June 25, 1979.
13. Heath, J., R., Carrol, B., D., and Cwik, T., T., "An Improved Version of CDL for the Efficient Simulation of Microprocessor and Minicomputer Systems," Journal of

Design Automation and Fault-Tolerant Computing, vol 2, no. 2, May 1978.

14. Hartenstein, R., W., and von Puttkamer, E., "KARL A Hardware Description Language as Part of a CAD Tool for LSI," Proceedings of the 4th International Symposium on Computer Hardware Description Languages, IEEE Computer Society, October 8, 1979.
15. Rome Air Development Center RADC-TR-78-6, Reconfigurable Computer Systems Design Facility, Initial Design Study, by D. R. Anderson, L. D. Anderson and K. Y. Wen, January, 1978.
16. Riley, R. P., Control System Design Language Implementation of a Gas Turbine Starting Controller, M.S. Thesis, Naval Postgraduate School, March 1984.
17. Ceital, A. J., Implementation of a Intel 8086 Base Realization Library for the Computer Systems Design Environment, M.S. Thesis, Naval Postgraduate School, June 1984.
18. Walden, H. J., Application of a General purpose DBMS to Design Automation, M.S. Thesis, Naval Postgraduate School, December 1983.
19. Ross, A. A., Computer Aided Design of Microprocessor-Based Controllers, Ph.D. Thesis, University of California, Davis, 1978.
20. Pollock, G. G., Further Development and Investigation of Computer-Aided Design of Microprocessor Systems, M.S. Thesis, University of California Davis, December 1981.
21. Walden, H. J., Application of a General Purpose DBMS to Design Automation, M.S. Thesis, Naval Postgraduate School, December 1983.
22. Riley, R. P., Control System Design Language Implementation of a Gas Turbine Starting Controller, M.S. Thesis, Naval Postgraduate School, March 1984.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3. LTC Alan Ross, Code 52RS Naval Postgraduate School Monterey, California 93943	4
4. Prof. Herschel H. Loomis, Code 62LM Naval Postgraduate School Monterey, California 93943	1
5. MAJ Theodore J. Smith, Jr. USA Computer Systems Acquisition Agency 200 Stoval Alexandria, VA 22331	1

208572

Thesis

S6052 Smith

c.1

Implementation of a
Zilog Z-80 base reali-
zation library for the
computer systems design
environment.

208572

Thesis

S6052 Smith

c.1

Implementation of a
Zilog Z-80 base reali-
zation library for the
computer systems design
environment.



thesS6052

Implementation of a Zilog Z-80 base real



3 2768 002 00791 6

DUDLEY KNOX LIBRARY